Learning from Cluster Examples

KAMISHIMA, Toshihiro

Doctor Thesis of Informatics, Kyoto University

2001

Abstract

Learning from cluster examples (LCE) is a hybrid task combining features of two common classification tasks: clustering and learning from examples. In LCE, each example is an object set with the true partition for the set, where the true partition is the one that users consider as the most appropriate for their aim among the possible partitions. The task is then to acquire a rule for partitioning unseen object sets from this example set. A method for learning such partitioning rules is useful in any situation where explicit algorithms for deriving partitions are hard to formalize, but where individual examples of true partitions are easy to specify. Clustering techniques have been of necessity applied to such situations, despite being essentially unsuited to the problems. We point out faults in using clustering techniques under such a situation, and explain why the techniques for LCE task expected to be overcome these faults. We then present a solution technique for LCE task, and apply the method to the problems in two domains; one with dot patterns and the other with more realistic vector-data images.

Contents

Chapter 1

Introduction

Clustering is a typical task that involves partitioning a given object set into subsets whose constituents are mutually similar. Since clustering is carried out based on rules or criteria given in advance, it can be regarded as *deductive* technique for partitioning.

In this paper, we advocate the use of an *inductive* technique for partitioning. In other words, we try to acquire a partitioning rule from an example set consisting of pairs of an object set and the true partition for the object set, where the true partition is the one that users consider as the most appropriate for their aim among the possible partitions. The acquired rule can then be used for finding the true partitions for unseen object sets (not appearing in the example set). Our induction task is similar to that of learning from examples, that acquires a rule for classification from a given example set, except that an aim of our task is to acquire a rule not for classification but for partitioning. Since our learning task also deal with partitioning like a clustering task, we give our new task the

composite name *learning from cluster examples*, or LCE.

A solution technique for LCE will be useful for any problem where users can easily identify which partition is the true partition for a given object set, but cannot specify explicit rules for deriving these partitions. No technique that has been developed for this aim. To fill the void, clustering techniques have been of necessity used, but they are not particularly suited to such kinds of partitioning. In this paper, we point out several faults caused by applying clustering techniques to such problems, and explain how our techniques are expected to overcome these faults.

We experimentally apply our technique to the problems for partitioning two types of data. We apply the method to the problems in two domains; one with dot patterns and the other with more realistic vector-data images. Since there are no other algorithms designed specifically for the tasks we consider, we cannot show direct comparison results. Therefore we pay particular attention to confirming whether our LCE algorithm has ability to acquire useful rules, and to analyzing the behavior of our method.

We proceed as follows. In Chapter 2, we show the importance of the LCE task. In Chapter 3, we formalize the problem. In Chapter 4 and 5, we then present partitioning and learning methods respectively. In Chapter 6, we explain experimental domains and a testing method. In Chapter 7, we show results and discuss them. Finally, Chapter 8 summarizes our conclusions.

Chapter 2

An Overview of Learning from Cluster Examples

In this chapter, we first present an overview of the LCE task, and then explain importance of the LCE task.

LCE is a composite task combining features from the techniques of clustering and of learning from examples. To give an overview of LCE, we therefore begin by reviewing these existing tasks.

Learning from examples is a task involving the acquisition of a rule for classification from a given example set. Each example is a pair of an object and a class to which the object should belong. The acquired rule is used to classify an unseen object into a proper class. The typical technique for this task in the machine learning field is ID3 [20] or feed-forward neural networks [2], and the task is often called discriminant analysis or pattern

recognition.

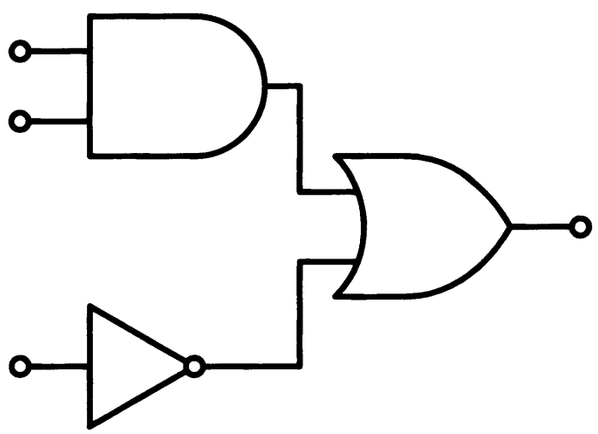
Clustering, on the other hand, is a task that partitions a given object set into clusters that have the properties of internal cohesion and external isolation [7]. The minimum distance or the k-means is a typical clustering method in the numerical taxonomy literature. In the machine learning literature, the task is often called *learning by observation* or *unsupervised learning*. COBWEB [8] and AutoClass [5] are typical examples of such a learning algorithm.

We have been developing "learning from cluster examples" techniques [12] as an extension of these two known approaches. The aim is not to find a rule to classify single objects, or a particular clustering, but to find a rule for partitioning, based on a given example set. Each example is a pair of an object set and an instance of the true partition for the object set. Note that, the true partition is the one that users consider as the most appropriate for their aim or intention. The acquired rule produced by learning from this example set is used to derive the true partition for an unseen object set. So, in contrast to learning from examples, LCE involves the acquisition of a rule not for classification but for partitioning an object set. And whereas the aim of clustering is to partition an object set based on rules or criteria given in advance, the aim of LCE is acquiring partitioning rule, that can be applied to any object set from the same domain. In short, LCE takes the inductive nature of learning from examples, and brings it to the task of clustering.

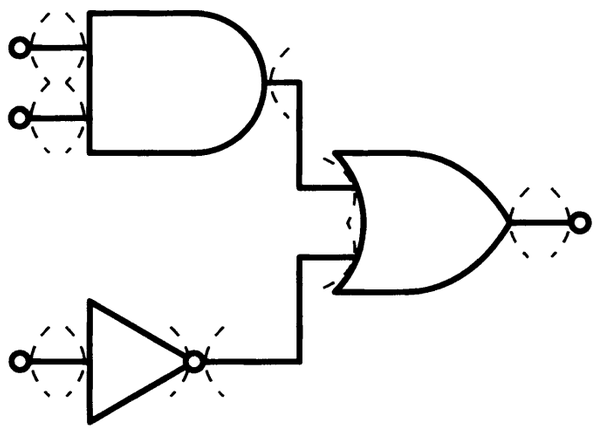
2.1 Why LCE Is Important

We will now describe some example cases which fit for the LCE task. Typically, these will be cases where an true partition for any object set is easy for a user to specify or identify, but where an overall set of rules for finding these partitions is very hard for a user to specify concretely and explicitly. A prime example of such a problem is image segmentation. Suetens et al [27] quoted Kanade's view of the segmentation problem, that is *to obtain a segmentation which separates out semantically meaningful objects or parts of objects*.

To explain the image segmentation task, we give an example of a typical problem involving the understanding of diagrammatic images. Figure 2.1(a) shows an image of a logic circuit diagram. Understanding this image is to obtain a proper description of the form of its logic circuit. In this case, a proper description for the image would be the logic function "$a \cdot b + \bar{c}$." In a typical diagram image understanding process, the given image is first of all partitioned, so that each cluster depicts an individual primitive symbol. This partitioning operation is generally called *segmentation* in the machine vision literature and is a very common technique. An appropriate treatment of the image in Figure 2.1(a), for example, would be to partition it into clusters with each depicting one part of a logic circuit diagram. Such a partition is illustrated in Figure 2.1(b), where the the original image has been separated with thin broken lines. After segmentation, each cluster is mapped to its proper primitive symbol. From the set of mapped symbols, an image description can then be inferred.



(a) An Original Image



(b) A Partitioned Image

Figure 2.1: Examples of diagram images

Segmentation is needed in many types of image understanding processes. As in the example we gave above, it typically corresponds to the task of finding partitions that satisfy the users' aims in situations where the users themselves cannot specify general rules for deriving partitions. Although segmentation problems are frequently encountered, we don't know methods that squarely grapples with the problem. Image segmentation techniques, for example, are usually designed in a non-systematic manner, relying on the designers' experience and intuition. Though such a design approach has been used from the beginning of machine vision research, the resulting programs are usually restricted to processing images in limited domains. We can pinpoint a number of drawbacks that arise from this absence of a systematic approach:

- Segmentation methods commonly rely on the designers' intuition. An example of a successful image understanding process is OCR (Optical Character Reader) systems. These systems are capable of recognizing regions where characters are written in a given document image. For this specific purpose, the powerful segmentation technique XY-Tree [9] was developed. This exploits a very specific feature of document image analysis: there are always gaps between lines or between characters. Another example of structure in a domain is RoboCup [14], in which soccer games are played by AI-controlled robots. These robots have to use machine vision to understand the game, but structure is artificially introduced by using distinct colors to identify objects. For example, the ball is orange and the goals are either blue or yellow. These coloring regulations are a significant aid that the robots attempts

to locate objects.

In cases like these two examples, human designers can state rules describing how images should be partitioned by using image features. In practice, character regions can be extracted by finding gaps between lines or characters in document images, and robots can almost always detect a ball by locating an orange region in the camera image. However, this kind of feature is not common. For example, Minoh et al have worked on the segmentation of line-drawing images, in which structure is hard to find [17]. This work succeeded in extracting symbol candidates from line-drawing images by defining a set of complicated rules for the extraction of symbols in terms of groups of short line-segments surrounded by a loop. This rule was intuitively derived based on a great deal of knowledge regarding the domain of line-drawing images, properties of image processing and cognitive science. In order to find suitable rules in domains where there is no obvious and constant features, the designers have nothing but to rely on intuition in addition to very much effort and knowledge.

- Some features are hard to formalize in pragmatic domains. The features, adopted in the above successful domains, are usually obvious, and is relatively easy to be represented by formal rules. We call such types of features *typical features*. However, there exists unexpected and ambiguous features that have to be taken into account for segmentation. We call them *exceptional features*. We give an example of exceptional features in the above Minoh's work. It is a very frequent event that

a surrounding loop happens to be cut, and extraction of symbols will be failed by this event. Such events can be often caused, for example, by stains on an original diagram, quantization errors in scanning, or the effects of image processing. The designers therefore have to take into account these events, but it is not easy to identify the features that how and where these events will occur. Such features are just what we call exceptional features. (In Section 6.2, we give some practical examples of such features). In a pragmatic domain, even though designers notice that these events will occur and try to formalize the features of the events, it is difficult to formalize such features as concrete rules by hand.

- Segmentation rules require user tuning. The designers of a system will create rules that express the typical features of the input, but these features will almost always allow for some variation. Since the nature of these variations are too difficult to capture intuitively, designers usually have no choice but to leave adjustable elements in segmentation rules. When applying a segmentation rule to a new image, experience and knowledge of machine vision is required for the users of the rules. For example, Minoh's work on image segmentation requests users to specify a threshold value to judge the shortness of the line-segments. Thus users without experience of machine vision techniques will not able to apply these rules.

- Segmentation results are statistically instable. We will show two reasons for this. Firstly, it is difficult to enforce a strict distinction between training and testing examples, because partitioning rules are typically created by hand. The designers

will naturally seek to find the best segmentation rules by referring to not only their knowledges of domain but also to available images. Thus, if the human designers just glancing the test images, they unwillingly gain some information from these images. Thus, since it is not avoidable to essentially distinct testing and training images, the performance for unseen images will not be objectively and rigorously evaluated. This facts weaken statistical stability of results derived by acquired rules by hand.

Secondly, an amount of information used for generating partitioning rules is restricted. Even though thousands or millions of images are available, the designers can merely deal with restricted amount of information due to the limitation of human cognitive ability. This fact also lead to statistical instability.

Other drawbacks have also been noted by Pavlidis, who pointed out the difficulties in finding partitioning when using several kinds of image features [19]. We believe that the only way to counter all these drawbacks is abandoning the non-systematic design approach in favor of a more powerful general method. Our choice for this method is a design approach based on LCE.

LCE expected to overcome the above drawbacks of existing approaches as follows:

- With LCE the designers only have to provide *instances* of partitions; it is not required to explicitly identify features important for segmentation by depending on their intuition.

- A learning algorithm can acquire rules that fully represent the domain. By analogy with learning algorithms for the object classification task, we can also see that LCE should handle exceptional features. In object classification, attribute values assigned to objects are often changed by accident, yet algorithm for learning from examples can still acquire successful classification rules. We are confident that a similar approach (that is, acquiring a rule with stochastic techniques from an example set) will also be effective for acquiring rules for partitioning.

- Just as object classification algorithms can generate rules that can cope with variance in the input, LCE can generate segmentation rules that users will not need to tune, and thus knowledge of machine learning or of the domain is not required when applying rules.

- Finally, since the learning algorithms explicitly require a set of training examples and can be effectively isolated from exposure to the testing examples, performance can be fairly evaluated. Since segmentation rules are acquired not by hand but by statistical algorithms, an amount of information gained from a given examples are not restricted by human cognitive ability any longer. These two property enhance statistical stability.

The development of successful techniques for learning from cluster examples will contribute to the progress of research in any field involving the mapping of raw sensor signals to abstract notions of objects. We have discussed a number of example domains

already, and the technique may also be applicable to problems such as multistrategy learning [16], the data mining [1] and the identification of genes in DNAs [4]. The rest of this paper will therefore rise to this challenge by presenting our algorithm for LCE.

Chapter 3

Formalization of Learning from Cluster Examples

This chapter formally states the task of learning from cluster examples. This task can be visualized as in Figure 3.1 and consists of two major stages: a learning stage and a partitioning stage. In the learning stage (Figure 3.1, left), the rule for carrying out partitioning is acquired from an example set. The example set, EX, includes $\#EX$ elements, $\{(O_1 , \pi_1^*) , (O_2 , \pi_2^*) , \ldots , (O_{\#EX} , \pi_{\#EX}^*)\}$, where O_I is an object set and π_I^* is an instance of its true partition. The object set O includes $\#O$ elements, $\{o^1 , o^2 , \ldots , o^{\#O}\}$. The cluster C^J is a subset of O, and the partition is a set of these clusters with $\#\pi$ elements, $\{C^1 , C^2 , \ldots , C^{\#\pi}\}$, such that the clusters are disjointed and every object has to be an element of exactly one of the C^J's. In the partitioning stage (Figure 3.1, right), based on the acquired rule, the true partition of an unseen object set, $O_{I'}$, is estimated.

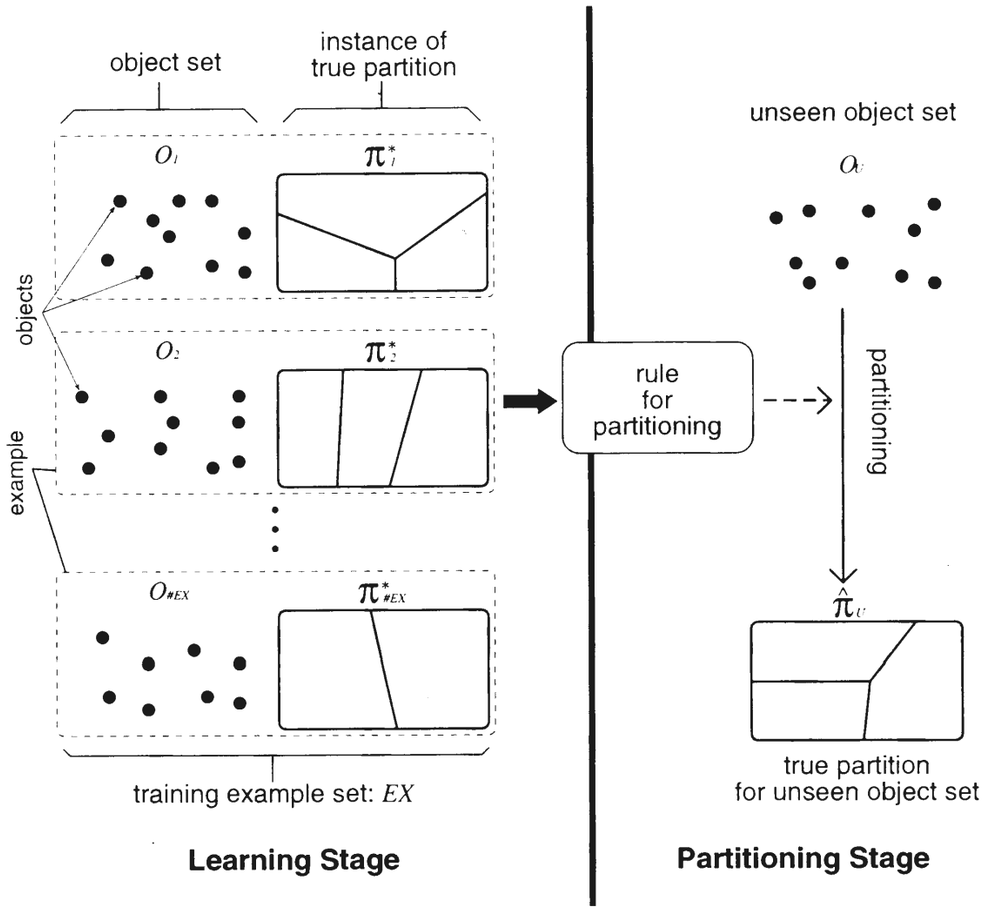


Figure 3.1: An illustration of learning from cluster examples

Because many of the algorithms used in techniques for learning from examples adopt attribute vectors to represent the individual objects, we also adopt them to represent the individual object set. We introduce the following three types of attributes assigned to different parts of the object set.

Attributes of Objects This type of attribute is assigned to constituent objects. For example, the positions of objects can be represented. We denote the attributes of the object o by $A(o)$. $A(o)$ is a vector with $\#A(o)$ values, $(a^1(o), a^2(o), \ldots, a^{\#A(o)}(o))$.

Attributes of Pairs This type of attribute is assigned to pairs of constituent objects. For example, the distances between object pairs can be represented. Specifically, let a pair of objects o^i and o^j be denoted by p^{ij}, and let P be the set of all possible pairs of objects. Thus, P has $\#O(\#O+1)/2$ elements, and this number is denoted by $\#P$. We denote the attributes of the object pair p by $A(p)$. $A(p)$ is a vector with $\#A(p)$ values, $(a^1(p), a^2(p), \ldots, a^{\#A(p)}(p))$.

Attributes of Partitions This type of attribute represents characteristics of entire partitions. For example, the number of clusters can be represented. While values of the above two types of attributes are rely on only an given object set, those of partitions are not. Given a partition, values of attributes of partitions are calculated from values of the above two types of attributes and from a states of the partition. When O is divided into a partition, π, we denote this type of attribute by $A(\pi)$. $A(\pi)$ is a vector with $\#A(\pi)$ values, $(a^1(\pi), a^2(\pi), \ldots, a^{\#A(\pi)}(\pi))$.

To apply our learning algorithm, the domains of attributes of objects and of pairs are either continuous numbers or discrete values (as in Quinlan's ID3 [20]). Also, the domains of attributes of partitions are real numbers from the interval $[0\ .\ 1]$.

3.1 Notes on The Formalization of LCE

We add here two notes relevant to the above formalization.

Firstly, though either attributes of objects or those of pairs are adopted to represent object sets when applying traditional clustering techniques, we adopt both types of attributes together in our formalization of the LCE task. Below we describe the reason why both types of attributes are adopted.

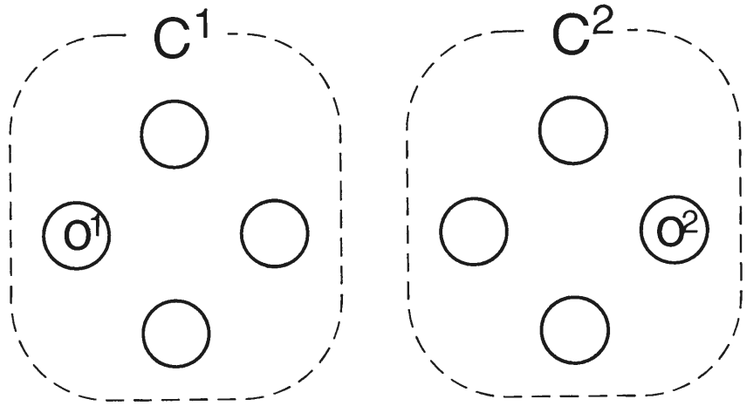
It is helpful to sort out partitioning tasks before showing explanation of the above reason. We suppose that the partitioning tasks can be classified into two categories, and call each of these *class finding* and *true clustering* respectively.

In the case of the class finding task, objects are independently generated from a population according to an identical distribution, and these generated objects compose an object set. In the population, there is a set of classes, and each object belongs to one of the classes. One can observe object sets themselves, but cannot do classes of the constituent objects. An aim of the task is to partition a given object set into clusters, each of which consists of objects belonging to the same unobserved class. For finding the true partition, it is therefore enough to investigate relations between features of each object and class properties. On the other hand, in the case of true clustering task, object sets are

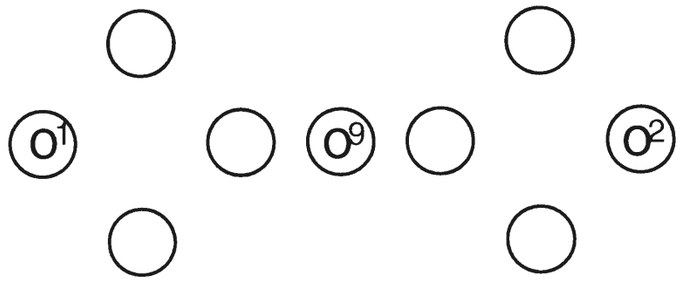
generated as a group. Constituent objects are not independent any longer, and the true partition is determined based on properties of entire the object set. Therefore, to derive the true partition, mutual influences among the objects have to be taken into account.

The following phenomenon clarify the difference between these two types of tasks. Figure 3.2(a) shows an object set O_1 that is generated from a population (objects are represented by circles). The true partition for the set consists of two clusters, C^1 and C^2, each of which is depicted by a surrounding broken line. Objects, o^1 and o^2, belong to clusters, C^1 and C^2, respectively. Consider then another object set, O_2 (shown in Figure 3.2(b)), that is identical except for the object, o^9. The object set is also generated from the same population. Examining the true partition for the set O_2 reveal distinction of two types of partitioning tasks. In the case of the class finding task, o^1 and o^2 are sure to belong to different clusters even in the O_2. Because objects are generated independently, the existence of the object o^9 will not affect weather o^1 and o^2 are in the same cluster or not. In the case of the true clustering task, these two objects might belong to the same cluster. This is because the mutual influences between the o^9 and the other objects might completely change the true partition for O_2.

To accomplish the class finding task, referring attributes of objects is sufficient for finding relations between individual objects and the unobserved classes. Therefore clustering techniques that only based on attributes of objects can be regarded as being designed for the class finding task. We may say that some of the clustering or unsupervised learning techniques, such as k-means or the AutoClass, are classified into this type of



(a) an object set, O_1, for which the true partition consists of two clusters.



(b) an object set, O_2, that is identical to the above set except for the object, o^9

Figure 3.2: Two examples of object sets to explain distinction between two types of partitioning tasks

clustering tasks. In contrast, to carry out the true clustering task, it is required to refer mutual influences among objects. So attributes representing features of such influences, i.e. attributes of pairs, is required. The so-called natural clustering tasks or the image segmentation tasks are typical examples. Clustering techniques that can handle attributes of pairs can be regarded as being designed for this type of task, and the minimum distance method is a representative of such techniques.

If a LCE technique can only deal with either of the two tasks, users have to specify which types of tasks they try to solve. Thus we consider that it is required to define LCE formalization that can acquire rules to applicable to the above both types of tasks. It is the reason that we employ both types of attributes together.

We then mention about using attributes of partitions. To derive the true partitions, one has to take into consideration not only the local features of object sets but also global features, i.e. attributes of partitions. For example, the attribute "the numbers of clusters", is typical example of such global features. To solve the image segmentation problem, it is required that proper numbers of clusters have to be specified automatically. If LCE techniques cannot deal with such global features, one will not able to apply the techniques to solve the segmentation problem. Therefore, we introduced attributes of partitions to our LCE formalization.

Chapter 4

The Partitioning Method

In this chapter, we describe our partitioning methods. In general, a partitioning rule is firstly learned and then the rule is applied for partitioning, but we describe partitioning method in this chapter for convenience of explanation.

Let π be an arbitrary partition for an unseen object set O, and $\pi=\pi^*$ be the event that the π is equals to the true partition π^*. To select the most plausible true partition among possible partitions, we adopt a maximum a posteriori (MAP) estimator, namely, the one that maximizes the joint probability of an event $\pi=\pi^*$ and all the attribute value vectors assigned to the object set. The joint probability is

$$\Pr[\pi=\pi^*, A(\pi): \{A(o)\}, \{A(p)\}], \tag{4.1}$$

where $\{A(o)\}$ and $\{A(p)\}$ are sets of all attribute value vectors assigned to constituents of O and P, respectively. Since $\{A(o)\}$ and $\{A(p)\}$ only depend on the given object

set and are independent from selection of π, we treat these value vectors as precondition to determine a distribution of the joint probability. Equation (4.1) is hard to calculate directly because a number of elements ($\#O+\#P+1$) is not constant, and this property is not suitable for most of statistical techniques. We therefore decompose it into the product of two terms and try to calculate each individually:

$$\Pr[\pi=\pi^*: \{A(o)\}, \{A(p)\}], \tag{4.2}$$

$$\Pr[A(\pi)|\pi=\pi^*: \{A(o)\}, \{A(p)\}]. \tag{4.3}$$

Maximizing the product of these two equations is the key to our method. To maximize Equation (4.2), we show how it can be manipulated into a more manageable form. The details of this manipulation are complicated and we defer them until the next section. Without going into the details of the representation here, the rewritten equation looks like:

$$\prod_{p\in P^+} f_1(p) \times \prod_{p\in P^-} \bar{f}_1(p). \tag{4.4}$$

As for Equation (4.3), we make the assumption that it is free from the preconditions $\{A(o)\}$ and $\{A(p)\}$. By definition, the value vector $A(\pi)$ is calculated from the vectors, $\{A(o)\}$ and $\{A(p)\}$, together with states of a partition, π. Therefore, the effects of $\{A(o)\}$

and $\{A(p)\}$ are already embedded in $A(\pi)$, even if we didn't explicitly refer to them as function preconditions. By introducing the assumption, Equation (4.3) is rewritten simply as the probability density:

$$\Pr[A(\pi)|\pi=\pi^*]. \tag{4.5}$$

This density is calculated by the function $f_2(A(\pi))$, which is acquired by the learning method described in Section 5.2. Consequently, to maximize Equation (4.1), all that we need to do is to maximize the product of Equation (4.4) and Equation (4.5).

We then describe our procedure to search for the most plausible true partition, that is achieving the maximum of the above product. According to the literature (e.g., [7]), the number of possible partitions for O is

$$\sum_{j=1}^{\#O}\left(\frac{1}{j!}\sum_{i=0}^{j}(-1)^{j-i}\binom{j}{i}i^{\#O}\right),$$

and this number increases exponentially according to the number of objects. Therefore, finding the optimal partition is not tractable in general, and we rely on the greedy search algorithm of Figure 4.1 to find a partition that may be locally optimal. In this algorithm, an initial partition is iteratively changed by applying modification operations. In each iteration, the operation that maximize the product of Equation (4.4) and (4.5) is applied. This iteration stops when no operation improves the product.

the procedure MAIN
$t := 0, \pi^0 := \{C = \{o\} \mid {}^\forall o \in O\}$
if (Eq5(π^0) > 0) then {
  $f :=$ true, $E^0 :=$ Eq5(π^0) $\times$ Eq4(π^0)
} else {
  $f :=$ false, $E^0 :=$ Eq4(π^0)
}
start:
  $t := t + 1, E^t := E^{t-1}$
  forall ($C^A \in \pi^{t-1}, C^B \in \pi^{t-1}, C^A \neq C^B$) {
    $\pi' := \pi^{t-1} - C^A - C^B + \{C^A \cup C^B\}$, call EVALUATION($\pi'$)
  }
  if ($f =$ true) {
    forall ($C^A \in \pi^{t-1}, C^B \in \pi^{t-1}, C^A \neq C^B$) {
      forall ($o \in C^A$) {
        $\pi' := \pi^{t-1} - C^A - C^B + \{C^A - \{o\}\} + \{C^B \cup \{o\}\}$, call EVALUATION($\pi'$)
      }
    }
  }
  if ($f =$ false $\vee E^t \neq E^{t-1}$) then goto **start**
output π^{t-1}
end

the procedure EVALUATION(π')
if ($f =$ false) then {
  if (Eq5(π') > 0) then {
    $f :=$ true, $\pi^t := \pi'$, $E^t :=$ Eq5(π') $\times$ Eq4(π')
  } else if (Eq4(π') $> E^t$) then {
    $\pi^t := \pi'$, $E^t :=$ Eq4(π')
  }
} else if (Eq5(π') $\times$ Eq4(π') $> E^t$) then {
  $\pi^t := \pi'$, $E^t :=$ Eq5(π') $\times$ Eq4(π')
}
return

Figure 4.1: Our algorithm for searching an true partition

The details of this algorithm is as follows. In Figure 4.1, Eq4(π) and Eq5(π) denote the values of Equation (4.4) and (4.5) when O is partitioned into π, respectively. The algorithm begins by creating an initial partition whose constituent clusters are made up of only one object, and then refines this partition so as to maximize the product of Equation (4.4) and (4.5). This refinement is done by applying two types of operations: a *merge*, that merges a pair of clusters, and a *move*, that moves one element from one cluster to another. When no partition that achieves a larger value of the product is found, this algorithm stops and then outputs the current partition as the most plausible true partition. Note that, the basic role of the procedure EVALUATION is to calculate a value of the product. The value is used to compare two partitions, one is the current, and the other is the one into which the current is transformed by applying arbitrary operations. The EVALUATION procedure treats separately the condition, where Equation (4.5) has been zero from the beginning of the algorithm, because the product becomes zero even if a value of Equation (4.4) is non-zero. Therefore, while this condition holds, EVALUATION simply returns the value of Equation (4.4), and the moving operation is not applied to avoid infinite loop. Once a partition for which Equation (4.5) is not zero is found, this special case is no longer invoked.

4.1 How To Maximize The Probability:

$$\texttt{Pr}[\pi{=}\pi^*; \{A(o)\}\ ,\ \{A(p)\}]$$

Here we give details on the transformation of Equation (4.2) into Equation (4.4) that we used above.

Because Equation (4.2) refers to many ($\#O{+}\#P$) value vectors as preconditions and the number of elements of these vectors is not constant, it is not straightforward to calculate its value. Therefore, we adopt the following technique to calculate it. We first generate a set of probabilities each of which is calculated based on two value vectors from $\{A(o)\}$ and one from $\{A(p)\}$. These probabilities are then combined by using Dempster & Shafer's rule of combination [26] (*DS rule* for short). So it is helpful to describe the DS rule before moving on to our calculation method for Equation (4.2).

The DS rule is used for combining probabilities based on different pieces of evidence. Let e be an event, E_a be an event set, and E_{All} be the set of all possible events. Let $P(E_{All})$ be the power set of E_{All}, i.e. $\{E_a : {}^{\forall}E_a \subseteq E_{All}\}$. $\texttt{Pr}[E_a]$ denotes the probability that one of the events in E_a occurs, and is called a *basic probability*. Basic probabilities satisfy these conditions:

$$\texttt{Pr}[E_a] \geq 0\ ,\ \texttt{Pr}[\emptyset] = 0\ ,\ \sum_{E_a \in P(E_{All})} \texttt{Pr}[E_a] = 1.$$

$\texttt{Pr}[E_a; A_x]$ denotes a basic probability for E_a based on the evidence A_x. Let $[E_a; A_x]$ be an event set for which $\texttt{Pr}[E_a; A_x]$ is defined. The difference between $[E_a; A_x]$ and E_a is

that the evidence A_x is given together or not. Given n distinct pieces of evidences, A_1 , ... , A_n, an combination of event sets, $\{[E_1; A_1]\ ,\ \ldots\ ,\ [E_n; A_n]\}$, is defined as follows. The $[E_1; A_1]$ is an arbitrary event set $E_1 \in P(E_{All})$ based on the evidence A_1. The rest of event sets are the same as $[E_1; A_1]$ except for that each of event sets is based on the distinct evidences $A_2\ ,\ \ldots\ ,\ A_n$. The $\{[E_1; A_1]\ ,\ \ldots\ ,\ [E_n; A_n]\}$ is a combination of these event sets. It should be noted that event sets, E_1 , E_2 , ... , E_n, may be different or identical. $\bigcap\{[E_1; A_1]\ ,\ \ldots\ ,\ [E_n; A_n]\}$ be the intersection of such a combination of event sets. According to the DS rule, the probability of a specific event e based on n evidences is

$$\frac{\sum_{\bigcap\{[E_1;A_1],\ldots,[E_n;A_n]\}=\{e\}}\left(\prod \texttt{Pr}[E_a; A_x]\right)}{1 - \sum_{\bigcap\{[E_1;A_1],\ldots,[E_n;A_n]\}=\emptyset}\left(\prod \texttt{Pr}[E_a; A_x]\right)}. \tag{4.6}$$

The numerator of the above equation denotes the sum of $\prod \texttt{Pr}[E_a; A_x]$'s in the case that $\bigcap\{[E_1; A_1]\ ,\ \ldots\ ,\ [E_n; A_n]\}$ is exactly equal to $\{e\}$ over the all possible combinations of event sets. $\prod \texttt{Pr}[E_a; A_x]$ denotes the product of basic probabilities assigned to $[E_a; A_1]$, ... , $[E_a; A_n]$ where $[E_a; A_x]$'s are the event sets that satisfy the condition of the sum. Concretely, consider a combination of event sets $\{[E_1; A_1]\ ,\ \ldots\ ,\ [E_n; A_n]\}$. If the intersection of the combination is equal to $\{e\}$, a product $\prod \texttt{Pr}[E_a; A_x]$ is $\texttt{Pr}[E_1; A_1] \times \texttt{Pr}[E_2; A_2] \times \cdots \times \texttt{Pr}[E_n; A_n]$. The numerator is the sum of products in all cases that a condition $\bigcap\{[E_1; A_1], \ldots, [E_n; A_n]\} = \{e\}$ is satisfied. The denominator is calculated in the same way except for summing in the case that the intersection of event sets becomes

an empty set.

We then present how to use the DS rule in our transformation of Equation (4.2). Strictly speaking, the presumptions and semantics of the probabilities manipulated by the DS and the Bayesian theories are different. However, it is well known that the DS theory can be regarded as a generalization of the Bayesian theory. Therefore, we introduce the DS theory to calculate the probability of Equation (4.2). Since the precondition of Equation (4.2) presents an event that sets of attribute value vectors $\{A(o)\}$ and $\{A(p)\}$ are simultaneously observed, by definition, the precondition corresponds to an event that $\{A(o)\} \cup \{A(p)\}$ are observed. This precondition can be treated as evidences in the context of the DS rule, because both of these can be regarded as basis to determine the distribution of $\pi{=}\pi^*$. An overview of the procedure to calculate the probability of Equation (4.2) is as follows: We first extract subsets from $\{A(o)\} \cup \{A(p)\}$ such that the union of the subsets exactly equals to $\{A(o)\} \cup \{A(p)\}$. We calculate basic probabilities whose evidences are each of the subsets, then combine these probabilities by applying the DS rule. The combined probability can be regarded the probability whose evidence is the attribute value set, $\{A(o)\} \cup \{A(p)\}$, since the union of the subsets equals to the set itself. As the subset of value vectors, we choose the subset, $\{A(o^i), A(o^j), A(p^{ij})\}$, that consists of attribute values related to an object pair, p^{ij}. Let $\mathsf{in}(p^{ij}, \pi)$ be the function that takes 1 if both o^i and o^j are in the same cluster of the partition π, and 0 otherwise. The following

$$
\begin{array}{lll}
\pi^1 = (o^1, o^2, o^3, o^4) & \pi^2 = (o^1)(o^2, o^3, o^4) & \pi^3 = (o^2)(o^1, o^3, o^4) \\
\pi^4 = (o^3)(o^1, o^2, o^4) & \pi^5 = (o^4)(o^1, o^2, o^3) & \pi^6 = (o^1, o^2)(o^3, o^4) \\
\pi^7 = (o^1, o^3)(o^2, o^4) & \pi^8 = (o^1, o^4)(o^2, o^3) & \pi^9 = (o^1)(o^2)(o^3, o^4) \\
\pi^{10} = (o^1)(o^3)(o^2, o^4) & \pi^{11} = (o^1)(o^4)(o^2, o^3) & \pi^{12} = (o^2)(o^3)(o^1, o^4) \\
\pi^{13} = (o^2)(o^4)(o^1, o^3) & \pi^{14} = (o^3)(o^4)(o^1, o^2) & \pi^{15} = (o^1)(o^2)(o^3)(o^4)
\end{array}
$$

Figure 4.2: An example of all the possible partitions for the object set: $\{o^1, o^2, o^3, o^4\}$

probabilities, $f_1(p^{ij})$, are calculated for each object pair in P:

$$f_1(p^{ij}) \equiv \mathtt{Pr}[\mathsf{in}(p^{ij}, \pi^*) = 1; A_C(p^{ij})],$$

where π^* is the true partition. The attribute $A_C(p^{ij})$ is a combination of the three attributes, $A(o^i)$, $A(o^j)$ and $A(p^{ij})$, which we will fully explain in Section 5.1. The function $f_1(p)$ is acquired in advance from an example set in the learning stage.

To compute Equation (4.2) by using the DS rule to combine the above probabilities, we first introduce some notations. The function $\mathtt{ev}(\Pi)$ is defined as $\mathtt{ev}(\Pi) = \{\pi{=}\pi^* : {}^\forall\pi \in \Pi\}$, where Π is an arbitrary set of partition. We use Π_{All} to denote the set of all possible partitions for O. Now, let us focus on the set of basic probabilities whose evidence is $A_C(p)$ of an arbitrary p. The function $f_1(p)$ can then be rewritten as the following basic probabilities:

$$
\begin{cases}
\mathtt{Pr}[\mathtt{ev}(\Pi(p)); A_C(p)] \equiv f_1(p), & \text{where } \Pi(p) = \{\pi : {}^\forall\pi \in \Pi_{All}, \mathsf{in}(p, \pi) = 1\} \\
\mathtt{Pr}[\mathtt{ev}(\bar{\Pi}(p)); A_C(p)] \equiv 1 - f_1(p), & \text{where } \bar{\Pi}(p) = \Pi_{All} - \Pi(p).
\end{cases}
$$

To give an example here, consider the object set $O = \{o^1, o^2, o^3, o^4\}$. For this set,

Π_{All} is the set of fifteen partitions as shown in Figure 4.2, where the objects in parenthesis form one cluster. And the set $\Pi(p^{12})$ is $\{\pi^1, \pi^4, \pi^5, \pi^6, \pi^{14}\}$, since these five partitions are the only ones which satisfy a condition that o^1 and o^2 are in the same cluster. In general, an event that both objects of p are in the same cluster, by definition, corresponds directly to an event that O is partitioned into any of the partitions in $\Pi(p)$. We assign a probability of zero to any subset of Π_{All}, except the two sets $\Pi(p)$ and $\bar{\Pi}(p)$. As a result, a set of basic probabilities consists of two non-zero probabilities, $\mathtt{Pr}[\mathtt{ev}(\Pi(p)); A_C(p)]$ and $\mathtt{Pr}[\mathtt{ev}(\bar{\Pi}(p)); A_C(p)]$, and the zero probabilities assigned to any other event sets except for these two.

Such sets of basic probabilities can be drawn for every pair in P, and hence $\#P$ probability sets can be derived. Since we treat preconditions of probability as evidences, each of these probability set can be considered as a set of basic probabilities based on evidences, $A(o^i)$, $A(o^j)$, and $A(p^{ij})$. And union of these evidences, exactly equals to $\{A(o)\} \cup \{A(p)\}$. Therefore, the combination of these probability sets corresponds to the probability based on the evidence, $\{A(o)\} \cup \{A(p)\}$.

Using this technique, a process to maximize Equation (4.2) for an arbitrary π is as follows. According to Equation (4.6), the combined probability is

$$\frac{\sum_{\bigcap\{[E_a:A_C(p)],\forall p\in P\}=\{\pi=\pi^*\}}\left\{\prod \mathtt{Pr}[E_a; A_C(p)]\right\}}{1-\sum_{\bigcap\{[E_a:A_C(p)],\forall p\in P\}=\emptyset}\left\{\prod \mathtt{Pr}[E_a; A_C(p)]\right\}}. \tag{4.7}$$

For a fixed p, we choose an event set $\mathtt{ev}(\Pi(p))$ as E_a if $\mathsf{in}(p, \pi)=1$, and a set $\mathtt{ev}(\bar{\Pi}(p))$

otherwise. This procedure is repeated for all p in P. The intersection of these event sets exactly consists of one element, $\pi=\pi^*$, and any other combination of event sets does not derive any sets including the event $\pi=\pi^*$. This is because, for each p in P, $\pi=\pi^*$ is always an element of either $\mathtt{ev}(\Pi(p))$ or $\mathtt{ev}(\bar{\Pi}(p))$. Consequently, in order to find the combined probability, we should choose a basic probability $\mathtt{Pr}[\mathtt{ev}(\Pi(p)); A_C(p)]$, if $\pi=\pi^*$ is an element of $\mathtt{ev}(\Pi(p))$, and $\mathtt{Pr}[\mathtt{ev}(\bar{\Pi}(p)); A_C(p)]$ otherwise. The numerator of Equation (4.7) is represented as follows:

$$\prod_{p\in P^+} \mathtt{Pr}[\mathtt{ev}(\Pi(p)); A_C(p)] \times \prod_{p\in P^-} \mathtt{Pr}[\mathtt{ev}(\bar{\Pi}(p)); A_C(p)],$$

where P^+ is a subset of P consisting of pairs that satisfy the condition $\mathsf{in}(p, \pi) = 1$, and P^- is its complementary set. For example, in the case of Figure 4.2, for the partition π^4, P^+ would be $\{p^{12}, p^{14}, p^{24}\}$. By introducing the function $f_1(p)$, the above equation can be rewritten as

$$\prod_{p\in P^+} f_1(p) \times \prod_{p\in P^-} \bar{f}_1(p), \tag{4.4'}$$

where $\bar{f}_1(p)$ is $1 - f_1(p)$. Looking once more at the example of Figure 4.2, the probability assigned to π^4 in this example would be:

$$f_1(p^{12})f_1(p^{14})f_1(p^{24}) \times \bar{f}_1(p^{13})\bar{f}_1(p^{23})\bar{f}_1(p^{34}).$$

The denominator of Equation (4.7) has the useful property of being constant for any possible partition. This is because the combination of event sets whose intersection becomes an empty set is independent of the choice of π. Therefore, since Equation (4.2) is proportional to Equation (4.4), the maximization of Equation (4.2) can be achieved by just maximizing Equation (4.4). Thus, to achieve our overall goal of maximizing the joint probability expressed in Equation (4.1), we can maximize:

$$f_2(A(\pi)) \times \prod_{p \in P^+} f_1(p) \times \prod_{p \in P^-} \bar{f}_1(p). \tag{4.8}$$

We add here a comment on the denominator of the combined probability. Calculating the value of this expression requires examining the condition where the intersection of the event sets becomes an empty set. This occurs only when there is a contradiction among the event sets. For example, consider an object set that consists of three objects, o^1, o^2 and o^3. For the set, if one observed an event that p^{12} and p^{13} is in the same cluster, one never observe an event that p^{23} is not in the same cluster. Thus the intersection of combination of event sets, $\mathrm{ev}(\Pi(p^{12}))$, $\mathrm{ev}(\Pi(p^{13}))$ and $\mathrm{ev}(\bar{\Pi}(p^{23}))$, becomes an empty set, and the probability assigned to this combination, $f_1(p^{12})f_1(p^{13})\bar{f}_1(p^{23})$, is adopted as a term of the denominator. There are two more combinations that lead to such a contradiction, and so the denominator becomes:

$$1 - \left(\bar{f}_1(p^{12})f_1(p^{13})f_1(p^{23}) + f_1(p^{12})\bar{f}_1(p^{13})f_1(p^{23}) + f_1(p^{12})f_1(p^{13})\bar{f}_1(p^{23}) \right).$$

In brief, the role of the denominator is to normalize the combined probability by eliminating the probabilities assigned to the combinations of events that lead to such a contradiction.

Chapter 5

The Learning Methods

In this chapter, we present the method for acquiring the two functions, $f_1(p)$ and $f_2(A(\pi))$ from the given example set, EX, in the learning stage.

5.1 Acquisition of The Function: $f_1(p)$

As described in the previous chapter, $f_1(p)$ is defined as:

$$f_1(p^{ij}) = \Pr[\mathsf{in}(p^{ij}, \pi^*) = 1 : A_C(p^{ij})].$$

This function is applied in two steps. At first, the value vectors, $A(o^i)$, $A(o^j)$, and $A(p^{ij})$, are combined into one value vector, $A_C(p^{ij})$. The function then derives the probability when the combined vector is given. The actual acquisition procedure of the function itself is also composed of two steps: a given training example set is first transformed into an

example set, ex_1, and then the function is acquired from this new set.

To acquire $f_1(p^{ij})$, it is required examples that are pairs of an observed value vector and a target value, i.e. $A_C(p^{ij})$ and $\mathsf{in}(p^{ij}, \pi^*)$ (a common format for the technique of learning from examples). We therefore transform a given example set EX into a set of examples in this form. Each example is generated from an object pair in an object set from the original example set. Thus, the number of elements in the transformed example set is the sum of the object pairs in the training example set, i.e. $\#ex_1 = \sum_{I=1}^{\#EX} \#P_I$. We denote a transformed example by $(A_C(p^{ij}), c)$, where the objects and object pairs are assumed to come from the same example (O_I, π_I^*). And where the class c takes the value $\mathsf{in}(p^{ij}, \pi_I^*)$, so the c becomes 0 or 1. The value vector $A_C(p^{ij})$ is calculated by combining the three attributes $A(o^i)$, $A(o^j)$, and $A(p^{ij})$. Our combination procedure is defined so as to be invariant under the ordering of indices, so that the value of the combined attribute $A_C(p^{ij})$ is always equal to $A_C(p^{ji})$. To produce such combined vectors, we copy all the values of $A(p^{ij})$ into the top of the combined vector. Additional elements are then concatenated to this combined vector by considering, one by one, the elements of the original vectors $A(o^i)$ and $A(o^j)$. The s-th elements of these original vectors, $a^s(o^i)$ and $a^s(o^j)$, are merged and added to the combined vector according to the following rules:

- If these two s-th elements take continuous values, the smaller value is added as an element of the combined vector, and the larger value is added as the subsequent element. That is, if the smaller value is added to the combined value as the t-th element, the larger value would be added as the $(t+1)$-th element.

- If these two s-th elements take discrete values, the values are merged into one and added into the combined vector. If the number of possible values for the original attribute is d, the merged attribute can take one of a possible $d(d+1)/2$ values. For example, if the possible values are "yes" and "no", the merged value can take one of the values "yes–yes", "yes–no", or "no–no".

As an example, consider the value vector $A(p^{ij})$ with two elements, the first discrete and the second continuous, and the vectors $A(o^i)$ and $A(o^j)$ both of which are with two elements, the first continuous and the second discrete. Given the attribute values $A(p^{ij}) =$ (yes , 100), $A(o^i) =$ (50 , yes), and $A(o^j) =$ (10 , no), the combined attribute $A_C(p^{ij})$ would be (yes , 100 , 10 , 50 , yes-no).

5.1.1 Our Algorithm to Estimate The Function: $f_1(p)$

We next describe the algorithm to estimate the function $f_1(p)$ from the transformed example set, ex_1. The example set can be simply represented by a form of $\{(A_1, c_1), (A_2, c_2), \ldots, (A_{\#ex_1}, c_{\#ex_1})\}$, where $c_I = \{0, 1\}$ is a value of the function $\mathsf{in}(p, \pi^*)$, and $A_I = (a^1, a^2, \ldots, a^{\#A})$ denotes the combined attribute value vector $A_C(p)$. This algorithm finds the conditional probability function, $\Pr[c_U{=}1|A_U]$ for an unseen vector A_U.

Before turning to the acquisition method of the probability function, we first present the *decision lists* [31] used for representing the function. Let T be a term that is the conjunction of literals L. The literal L is a logical function that can take the binary values true or false when an attribute values a is given, as follows:

- for attributes a that take continuous values, the three possible forms of the literal are $(\theta_l \leq a)$, $(\theta_l \leq a < \theta_u)$, and $(a < \theta_u)$, where θ_l and θ_u are proper threshold values. Such a literal takes true whenever the value a satisfies the condition specified by the literal.

- for attributes a that take discrete values from some set V, a literal has the form $(a = v_1 \vee v_2 \vee \cdots \vee v_{a'})$, where $v_1, \ldots, v_{a'}$ are elements of the set V. This literal takes true whenever the value of a is one of $v_1, \ldots, v_{a'}$.

Decision lists are defined as a pairing of an ordered term list $\langle T_1, T_2, \ldots, T_{m-1}, \text{true}\rangle$ and a probability list $\langle \mathtt{Pr}_1, \mathtt{Pr}_2, \ldots, \mathtt{Pr}_m\rangle$, where true is a term that always outputs true. Specifically, when the unseen value vector A_U is applied to a term list in the order $T_1, T_2, \ldots$, true, if T_k is the first term that outputs true, the decision lists output the value of the corresponding $\mathtt{Pr}_k$ as the conditional probability $\mathtt{Pr}[c_U{=}1|A_U]$.

We note here the reason why we adopt not decision trees but decision lists. First, Pagallo and Haussler [18] have pointed out that the size of the decision trees tends to drastically increase if the concept to be learned is disjunctive. Secondly, the size of the example set drastically decreases, since the decision trees are usually acquired by a so-called divide-and-conquer procedure and the example set is divided whenever a new node is created. This property weaken stochastic stability.

Our algorithm for acquiring the above decision lists is described in the Figure 5.1. This algorithm finds the most probable decision list based on Rissanen's MDL (Minimum Description Length) principle [24, 25], which has been successfully adopted in learning

the procedure SEARCHING
example set$S := ex_1$
term no. i:=0, decision list $DL := \langle\rangle$
conditional probability $PR := \langle\rangle$
do {
  $i := i + 1$
  the number of updating times $j := 0$
  j-th updated term $T_i^j :=$ true
  do {
    $j := j + 1$
    Let L_B be the literal maximizing the evaluation function
      and G_B be the function value for the L_B †
    if $(G_B \leq 0)$ then {
      $T_i^j := T_i^{j-1}$, goto **term_end**
    }
    $T_i^j := T_i^{j-1} \wedge L_B$
  } until(every classes of elements in $S(T_i^j)$ is all 0 or all 1)
term_end:
  if ($T_i^j =$ true) then goto **list_end**
  Add T_i^j to DL and $\mathtt{Pr}(S(T_i^j))$ to PR ††
  $S := S - S(T_i^j)$
} until(every classes of elements in S is all 0 or all 1)
list_end:
Add true to DL and $\mathtt{Pr}(S)$ to PR

the procedure PRUNING
total code length $\ell := \ell(ex_1, DL)$
the number of terms $m := i + 1$
while$(m > 1)$ {
  $S' := S(T_m) \cup S(T_{m-1})$
  $DL' := \langle T_1, \ldots, T_{m-2}, \text{true}\rangle$, $PR' := \langle \mathtt{Pr}_1, \ldots \mathtt{Pr}_{m-2}, \mathtt{Pr}(S')\rangle$
  $\ell' = \ell(ex_1, DL')$
  if $(\ell \leq \ell')$ then goto **noprune**
  $DL := DL'$, $PR := PR'$, $m := m - 1$, $\ell = \ell'$
}
noprune:
output DL, PR
end

Figure 5.1: Our algorithm for searching decision lists

from examples techniques [15, 21, 29]. This principle selects the best model from a given set of candidate stochastic models and is stated as "select the model in the observed data that permits the shortest encoding both of the observations and the model." Grounded in this principle, we formalize a set of stochastic models representing the conditional probability functions and specify a coding scheme for this set. In Figure 5.1, we show the procedure for finding the decision list that permits the shortest code length, and the coding schemes of the decision lists are summarized in Appendix A. We here make some remarks related to Figure 5.1.

This algorithm is composed of two procedures: SEARCHING and PRUNING. The former is the procedure for finding a decision list by repeatedly adding terms so as to achieve the shortest code length and then removing examples satisfied by the list. In the latter procedure, the acquired decision list is polished.

We first discuss the evaluation function G_B and the literal L_B at the mark † in Figure 5.1. This evaluation function is designed to find the term that is useful for achieving the shortest code length. Let $S(T_1)$ be the subset of the current example set S that consists of elements that satisfy the condition specified by the term T_1. Let $\#S(T_1)$ be the number of elements in $S(T_1)$, and $\ell(S(T_1))$ be the code length for $S(T_1)$. Assume two terms T_1 and T_2 that satisfy the condition $S(T_1) \supseteq S(T_2)$. If the condition

$$\frac{\ell(T_1) + \ell(S(T_1))}{\#S(T_1)} > \frac{\ell(T_2) + \ell(S(T_2))}{\#S(T_2)}$$

is satisfied, the evaluation function is

$$G(T_1, T_2) = \big(\ell(T_1) + \ell(S(T_1))\big) - \Big(\ell\big(T_2\big) + \ell(S(T_2)) + \ell\big(S(T_1) - S(T_2)\big)\Big),$$

and otherwise 0, where $\ell(T_1)$ is the code length for T_1. The L_B is the literal that maximizes the evaluation function $G(T_i^{j-1}, T_i^{j-1} \wedge L)$ over all literals L that satisfy the condition $S(T_i^{j-1}) \supseteq S(T_i^{j-1} \wedge L)$. G_B is the output of the function at that time.

Next, we comment on $\mathtt{Pr}(S(T_i^j))$ calculated at the mark ††. Because we adopt the coding scheme of example sets in [29], this is defined as:

$$\mathtt{Pr}(S(T_i^j)) = \frac{\#S^+(T_i^j) + 1}{\#S(T_i^j) + 2},$$

where $S^+(T_i^j)$ is composed of the elements in $S(T_i^j)$ whose class labels are 1. Details about the code length of decision lists and example sets are shown in Appendix A.

We finally add comments on the reason why we adopt our original algorithm, despite many algorithms for estimation of posterior probabilities have been developed to date. These algorithms are designed so as to try to minimize the expected 0-1 loss, which is the ratio of incorrectly classified examples. For example, the method for acquiring decision tree based on MDL principle [15] using coding scheme dealing with the 0-1 loss. Most of existing algorithms designed for this purpose. In contrast to this, we introduce coding scheme aiming to minimize the KL divergence, which measures how closely the probability is estimated. Since, for the function used in the LCE task, it is important

that the ability can estimate probability as precisely as possible, we develop and use an algorithm having the property.

5.2 Acquisition of The Function: $f_2(A(\pi))$

We next describe the method for acquiring the function $f_2(A(\pi))$ that is required for the calculation of Equation (4.5). This function is the conditional probability density of $A(\pi)$ given the event $\pi{=}\pi^*$.

Our algorithm to derive the density function requires an input whose form is a set of attribute value vectors. Therefore we transform the original training example set, EX into this form. Recall that the set EX is composed of examples of object sets O_I with their true partitions π_I^*. For each element of this set, we calculate the attribute value vector $A(\pi_I^*)$. We refer to the set of these vectors as the transformed example set, ex_2. Since each element of the ex_2 follows the density, $\Pr[A(\pi)\ ,\ \pi{=}\pi^*]$, we can derive the $f_2(A(\pi))$ by estimating $\Pr[A(\pi)\ ,\ \pi{=}\pi^*]$ from the ex_2 and then dividing this by $\Pr[\pi{=}\pi^*]$. However, the $\Pr[\pi{=}\pi^*]$ cannot be estimated pragmatically. This is because the number of possible partitions is enormous in comparison to the number of given examples. Therefore we assume that $\Pr[\pi{=}\pi^*]$ is uniform, so the $\Pr[A(\pi)\ ,\ \pi{=}\pi^*]$ come to be proportional to $f_2(A(\pi))$.

Now, all we have to do is estimating $\Pr[A(\pi), \pi{=}\pi^*]$. With the set ex_2 as its input, our algorithm described below can calculate the density. We employ *regression trees* [3] to represent the density function. So before turning to explain our algorithm, it is helpful to

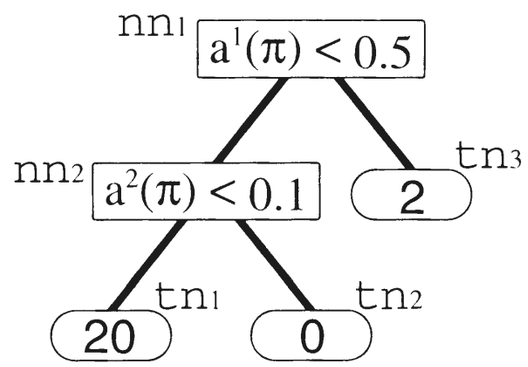


Figure 5.2: An example regression tree

describe the regression trees. An example of a regression tree is shown in Figure 5.2. The tree in this figure has terminal and non-terminal nodes. Each non-terminal node, shown by a rectangle, has one threshold, one index that specifies which element of $A(\pi)$ should be compared, and two branches connecting it to other nodes. In addition, each terminal node, shown by a rounded rectangle, has a probability density value. When a fixed value vector $A(\pi)$ is given, the proper probability density value is found by recursively descending through the regression tree to a terminal node, as follows. First, the vector is compared to the threshold specified at the root node of the tree (for the specific index indicated at the node). If the value is smaller than the threshold, then the left branch of the node is descended. Otherwise, the right branch is descended. If the next node in the tree is also non-terminal, the process of comparing the specified attribute value and the threshold at the node is repeated, until a terminal node is reached. At a terminal node, the proper probability density value is simply the value specified by the node. For example, suppose

that the vector $(0.3\ ,\ 0.5)$ is applied to the regression tree in Figure 5.2. The first value of the vector is compared to a threshold of 0.5 at the root node (labeled nn_1). Since the value is smaller than the threshold, the left branch is traced and the node nn_2 is found. The node nn_2 is also non-terminal, so the second value of the vector is then compared to the threshold 0.1. As a result, we reach the terminal node tn_2. This gives a value of 0 as the target probability density 0.

Thus, if we can compute a regression tree, we can find the probability density of partition attributes. To do this, we use the algorithm as follows. This procedure is also grounded in the MDL principle. We describe a set of stochastic models and define a scheme for coding both of the models and the given example set. Then, as the function $f_2(A(\pi))$, we employ the model that permits the shortest code length.

We here present the coding scheme for the regression trees, that are used for representing the target function. The code length for a structure of the regression tree equals the total number of nodes. The article [21] presents a full explanation of the code length and of the coding scheme for the tree. For each non-terminal node, a threshold and an index at the node must be encoded. The threshold is encoded in the same scheme as that used for the threshold of the decision lists in Appendix A, and the code length for the index is $\log \#A(\pi)$. Note that $\log$ is logarithm whose base number is 2 and $\ln$ denotes natural logarithm in this paper. The scheme presented here makes it feasible to specify an arbitrary regression tree, RT, with code length $\ell(RT)$. Next, an example set, ex_2, must be encoded by using this regression tree. According to [32], the total code length is

approximated

$$\ell(ex_2\ ,\ RT) = \ell(RT) + \{-\log \mathcal{L}(ex_2|RT) + \frac{1}{2}\#TN(\log e + \log \#ex_2)\}.$$

where $\mathcal{L}(ex_2|RT)$ is the likelihood, $\#ex_2$ is the number of examples in ex_2, and $\#TN$ is the number of terminal nodes in RT. The likelihood of ex_2 is defined as follows. Let TN be the set of all terminal nodes in the RT, and tn_x be its element. Let $\#\mathsf{tn}_x$ be the number of examples in ex_2 that reaches the terminal node tn_x. $\mathcal{L}(ex_2|RT)$ is defined as

$$\mathcal{L}(ex_2|RT) = \prod_{\mathsf{tn}_x \in TN} \mathtt{Pr}[\mathsf{tn}_x]^{\#\mathsf{tn}_x},$$

where $\mathtt{Pr}[\mathsf{tn}_x]$ is the probability density at the node tn_x defined as:

$$\mathtt{Pr}[\mathsf{tn}_x] = \frac{\#\mathsf{tn}_x}{\#ex_2 \times V(R(\mathsf{tn}_x))}.$$

$R(\mathsf{tn}_x)$ is the region for a value vector such that if the vector ranges $R(\mathsf{tn}_x)$, it reaches the terminal node tn_x, and $V(R(\mathsf{tn}_x))$ is the volume of $R(\mathsf{tn}_x)$. For example, in the case of node tn_2 in Figure 5.2, any value vectors within the range $a^1(\pi) < 0.5$ and $a^2(\pi) \geq 0.1$ that are inputted to this regression tree would reach the node tn_2. So $R(\mathsf{tn}_2)$ is $(0 \leq a^1(\pi) < 0.5) \wedge (0.1 \leq a^2(\pi) \leq 1))$, and $V(R(\mathsf{tn}_2))$ is $0.45 (= (0.5 - 0) \times (1 - 0.1))$.

In order to acquire the function $f_2(A(\pi))$, we must find the regression tree that permits the shortest total code length $\ell(EX_2\ .\ RT)$. For this purpose, we introduce an algorithm in

the procedure MAIN
$ex_2 := \{A(\pi_1)\ ,\ A(\pi_2)\ .\ .\ .\ .\ .\ A(\pi_{\#ex_2})\}$: the example set
RT := (TN , NN): a regression tree whose root node is terminal
  TN := $\{\mathsf{tn}_1\}$: a set of terminal nodes (tn_1 is a root node)
  NN := $\{\}$: a set of non-terminal nodes
for s from 1 to $\#A(\pi)$ {
  $\delta^s := 6 \times \langle$standard deviation of $a^s(\pi_1)\ ,\ a^s(\pi_2)\ ,\ .\ .\ .\ ,\ a^s(\pi_{\#ex_2})\rangle / \#ex_2$
}

start:
$\mathrm{RT}_{best} := \mathrm{RT}$
foreach tn′ in TN {
  for s from 1 to $\#A(\pi)$ {
    Let l and u be the lower and the upper bound
      of the s-th attribute of the region $R(\mathsf{tn}')$ respectively
    for d from 1 to ∞ {
      $q := (1/2)^d$
      if $(q < \delta^s)$ then goto **checkend**
      for t from 1 to $2^{(d-1)}$ {
        $\theta := q(2t-1)$
        if $(l \leq \theta < u)$ then {
          RT′ := (TN′ . NN′)
            NN′ := NN ∪ nn′
            (nn′ is the terminal node whose threshold is θ
              and is placed at the position used to be tn′)
            TN′ := $\{\mathrm{TN} - \mathsf{tn}'\} \cup \{\mathsf{tn}^R_{new}\ ,\ \mathsf{tn}^L_{new}\}$
            (tn^L_{new} and tn^R_{new} are the right and the left node of of the nn′)
          if $(\ell(ex_2\ ,\ \mathrm{RT}') < \ell(ex_2\ ,\ \mathrm{RT}))$ then RT := RT′
        }
      }
    }
checkend:
  }
}
if $(\mathrm{RT}_{best} \neq \mathrm{RT})$ goto **start**
end:
output RT_{best}
end

Figure 5.3: Our learning algorithm for searching regression trees

Figure 5.3. This algorithm adopts a divide-and-conquer strategy that recursively divides a given training example set. Initial regression tree consists of only one terminal node, and represents an uniform density function that is always at constant 1. The current tree is iteratively modified. This modification operation is as follows: One of terminal nodes of the current tree is replaced with new non-terminal node and two new terminal nodes are added at the branches of the new non-terminal node. The replaced terminal and the new non-terminal node are selected so as to maximize $\ell(EX_2\ ,\ RT)$. Finally, this algorithm stops when no improvement is feasible, and outputs the current tree.

Chapter 6

Experimental Domains and Testing Methods

We have applied our technique of learning from cluster examples to two test domains: dot patterns and vector-data images.

6.1 Experimental Domains: Dot Patterns

Segmentation of dot patterns is a basic problem for clustering that features dots scattered in a 2–dimensional space of the same width and height, as shown in Figure 6.1. Here, all the dots from the same cluster are depicted with the same type of symbols. These dots are usually generated by considering a number of circular regions that are also placed in the space (in the figure, these regions are depicted by dotted lines). These circles are used to create clusters by generating dots according to a Gaussian distribution (note that if any

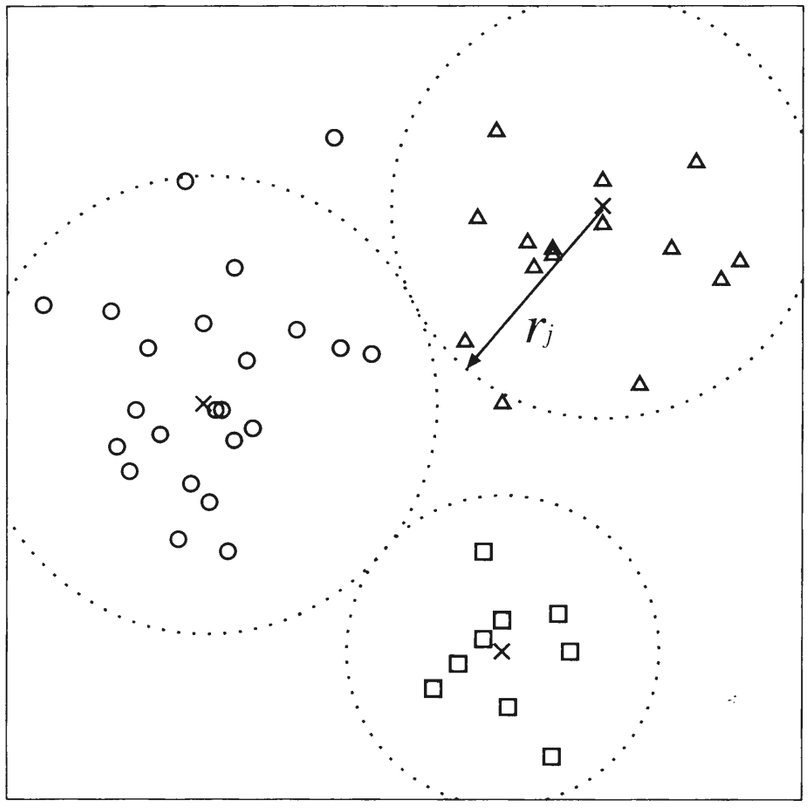


Figure 6.1: An example of a dot pattern

dot falls outside the 2–dimensional space in this generation process, it is discarded and a replacement is generated). For each region, the mean of the Gaussian distribution of its dots is at the center of the circle, and the we use the following 2-dimensional Gaussian mixture distribution:

$$f(x\,,\,y) \;=\; \sum_{j=1}^{m} z_j N(x\,,\,y;\,\mu_j\,,\,\nu_j\,,\,\sigma_j)\,,$$

$$N(x\,,\,y;\,\mu_j\,,\,\nu_j\,,\,\sigma_j) \;=\; \frac{1}{2\pi\sigma_j^2}\exp\left(-\frac{(x-\mu_j)^2+(y-\nu_j)^2}{2\sigma_j^2}\right),$$

where m is the number of clusters, z_j specifies the ratio of mixing, μ_j and ν_j are means of the x and y positions, and σ_j is a standard deviation. This standard deviation differs depending on the types of example sets. We prepared three types of example sets with varying degrees of overlap between the different clusters. To create each set, we first randomly generated a value for m, and created n random points within the space. We then created circular regions of radius r_j , $1 \leq j \leq m$ around each of these points, by generating r_j randomly under the constraint that each of the resulting circles must touch at least one other (as shown in the example of Figure 6.1). Depending on the type of example set we wanted to create, we then assigned σ_j to be either $\sigma_j = r_j/3.0$ (for a *separated* example set), $r_j/2.5$ (for a *touching* example set) or $r_j/2.0$ (for a *overlapping* example set). Note that we force the covariance to be zero, namely, the x and y deviations are equal.

Our three example sets each contain 10 to 1000 elements. Each object set in the example sets contains 50 dots composing two to four clusters. Four attributes of objects, eight attributes of pairs and one attribute of partitions are used, as detailed in Section 6.1.1.

To provide a comparison for our learning from cluster examples technique, we also applied the following EM algorithm (a common clustering technique) [6] to the task of partitioning the dot patterns. Let n be the number of observed objects and $(x_i\ ,\ y_i)$ be the position of the i-th object. The EM algorithm leads to the parameter values of a distribution function so as to maximize the log-likelihood:

$$\log \mathcal{L}(x_1\ ,\ \ldots\ ,\ x_n\ ,\ y_1\ ,\ \ldots\ ,\ y_n) = \sum_{i=1}^{n} \log f(x_i\ ,\ y_i).$$

After the parameter values are estimated, each object is classified into the k-th cluster such that:

$$k = \underset{j=1,\ldots,m}{\operatorname{argmax}}\ z_j \mathcal{N}(x_i\ ,\ y_i; \mu_j\ ,\ \nu_j\ ,\ \sigma_j).$$

The initial conditions we used for the EM algorithm parameters were as follows. First, we assumed that the correct number of clusters $\#\pi^*$ was explicitly given as m, and then we initialized all the σ_j's to $S/6$ (S is width or height of the 2–dimensional space) and all the z_j to $1/\#\pi^*$. As an initial guess at the actual clustering, we assumed that the means of the clusters were equi-distantly placed on a circle of radius $0.3 \times S$ in the center of the space.

6.1.1 Attributes for Dot Patterns

In this section, we give full explanation of attributes that is adopted in our experiments. We select these attributes by using the primitive feature selection technique as follows. First, we select widely used attributes as candidates. In this selection, we reject attributes that need laborious tuning or too much computation resources. We then extract many attribute subsets from the set of the candidate attributes. Finally, by using testing method described later, we adopt the best among these attribute subsets. Note that, it is certainly clear that selection of attributes will affect the performance of estimation. But this attribute selection problem is irrelevant to the main subject, and to follow up this matter would take us beyond the scope of this paper. Very many types of attributes have been developed in the vision literature, and the attribute selection problem is a major topic in the machine learning literature. We suppose that results obtained by these works will helps to solve our attribute selection problem.

The attributes assigned to objects and their pairs are briefly shown in Table 6.1. All the attributes of objects should be fairly self-explanatory, being the X and Y coordinates and the Euclidean distances to the k-th nearest neighbor (the value of parameter k will be introduced below) and to the nearest dot. As for the attributes of pairs, the first attribute is simply the Euclidean distance between two dots, but the second and the third attributes require the imposition of a total order on the dots. Let us call the dots in the dot pair A and B. First, the dots are ordered according to their Euclidean distance from dot A, and the position index number of dot B in this order is found. Then, the dots are ordered

Attributes of objects	Attributes of pairs
1. X–position 2. Y–position 3. Distance to the k-th nearest dot 4. Distance to the nearest dot	1. distance between two dots 2. The smaller position index number in ascending distance order 3. The larger position index number in ascending distance order 4. k-th nearest factor 5. Gabriel Graph factor 6. Relative Neighborhood Graph factor 7. Length of the longest edge on its MST path 8. The number of edges on the MST path joining the dots

Table 6.1: Attributes for dot patterns

according to their distance from dot B, and the position index number of dot A in this order is found. The second attribute of pairs then the smaller of these position index numbers, and the third is the larger.

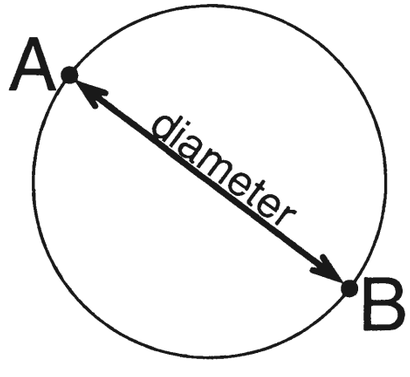
The fourth attribute of pairs is related to Wong and Lane's work [30]. They employ the following factor for clustering:

$$\frac{\#O}{2k}(V_k(A) + V_k(B)),$$

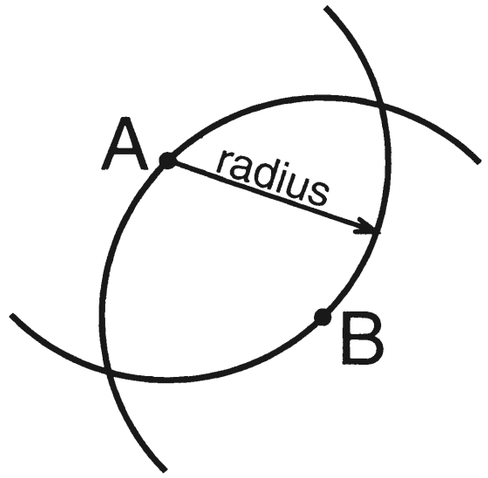
where k is an adjustable parameter. The function $V_k(\cdot)$ gives the volume of a region centered on a dot with radius r_k, the distance to the k-th nearest dot. In 2–dimensional space, this is simply $V_k(\cdot) = \pi {r_k}^2/2$. We employ Wong and Lane's factor as the fourth attribute of the dot pairs, with parameter k (also mentioned above in relation to the third attribute of objects) set to the value of $2 \ln \#O$, as suggested in [30].

The fifth and sixth attributes of pairs are related to Urquhart's work [28] on graph theoretical clustering. Urquhart proposes a clustering technique based on a *Gabriel graph* (GG) and a *relative neighborhood graph* (RNG). The GG is a graph having edges between two dots, A and B, if no other dot lies in the circular region that can be constructed between them as shown in Figure 6.2(a). The RNG is similar to this, except that the area considered between the two dots is the region described in Figure 6.2(b). We adopt the number of dots in these two types of regions as the fifth and the sixth attributes of pairs in Table 6.1.

The seventh and eighth attributes are related to Zahn's pioneering work on graph the-



(a) Region for a Gabriel Graph



(b) Region for a Relative Neighborhood Graph

Figure 6.2: Dot-free regions in Gabriel graphs and relative neighborhood graphs

oretical clustering [33] that employs a *minimal spanning tree* (MST). The MST is defined as a tree connecting all the dots in a given dot pattern for which the sum of the lengths of its constituent edges is minimal among all possible trees. There is only one path between any pair of two dots on an MST. For the dots A and B, we adopt the length of the longest edge on the MST path between them as the seventh attribute and the number of edges on this path as the eighth attribute.

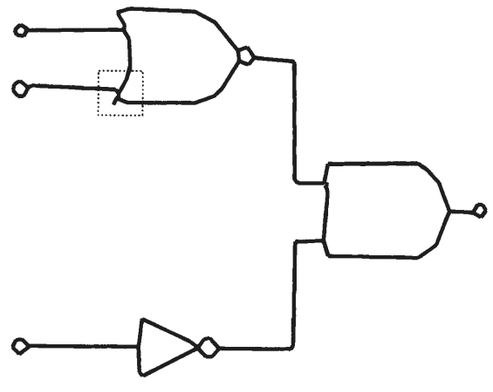
Finally, we adopt the following one attribute of partitions:

$$a^1(\pi) \equiv \frac{\#\pi}{\#O}$$

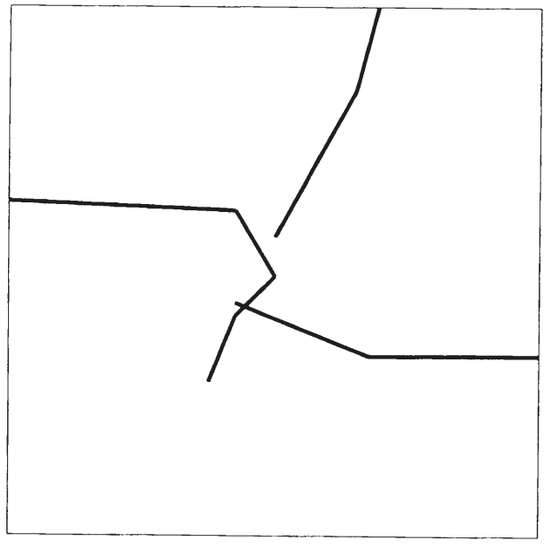
6.2 Experimental Domains: Vector-data Images

Vector-data images are often used in the process of diagram image understanding. A vector-data image is represented by using line-segments and is typically used to represent images drawn with thin lines, such as diagrams or maps. A line-segment is a straight line connecting two end-points. Each line-segment corresponds to an object, and an entire image does to an object set. Partitioning this type of object set is a more realistic task than dot pattern partitioning.

We generated our example set of vector-data images by transforming handwritten logic circuit diagrams. Handwritten diagrams were scanned by an image scanner, and the common image processing techniques of thinning and vectorization were then applied. The original handwritten diagrams consisted of five kinds of circuit parts: AND-gates,



(a) a whole image



(b) an enlarged image

Figure 6.3: Examples of vector-data images

OR-gates, buffers, terminals, and connecting lines. An example of a vector-data image of a logic circuit diagram is shown in Figure 6.3(a). The segmentation task is then to divide these vector-data images into clusters such that each cluster consists of line-segments whose origins are the same circuit part.

This example set consists of 100 elements. Over this example set, the mean number of clusters is 16.7, and the mean number of objects per one object set is 102.9. Eight attributes of objects, seven attributes of pairs, and four attributes of partitions are used, as shown in Section 6.2.1.

In Section 2.1, we pointed out four drawbacks of a non-systematic design approach, and we can see here that our vector-data set is a good test bed for each of these drawbacks. Firstly, in terms of the images themselves, there are no obvious features that help for partitioning the vector-data images. For instance, there are no visual clues such as contrived color or markings. Such partitioning problem is just what we want to solve without depending on intuition. Secondly, the difficulty of formalizing exceptional features can be illustrated by examining some of the possible events in the images. For example, Figure 6.3(b) is an enlarged image that depicts the highlighted part of the OR-gate symbol in Figure 6.3(a). This shows that there are several undesired events (*spurs* and *gaps*) in the image. The presence of such undesired events significantly complicates the task of specifying concrete rules, since it is difficult to identify the features that how and where these events will occur. We want our algorithm to be capable of acquiring concrete rules that can be applied to such images. Thirdly, since the original diagrams are hand-written,

these vector-data images are suffered some variation. The kind of solution that we want to find for partitioning these images is one that is free from user tuning when applying acquired rules. Finally, in order to enhance statistical stability of estimation, we also want to rigorously separate the training examples from the testing ones. And our algorithm deal with so many numbers of examples that exceeds the limitation of human cognitive ability. These are what we have achieved with our LCE technique, as demonstrated in the next chapter.

6.2.1 Attributes for Vector-Data Images

The attributes assigned to objects and their pairs are briefly shown in Table 6.2. The selection procedure of these attributes is the same in the case of experiments for dot patterns.

The first four attributes of objects are simply the X coordinate of the line-segment's mid-point, the Y coordinate of the mid-point, the difference between the X coordinates of the two end-points, and the difference between the Y coordinates of the two end-points.

All the other attributes of objects are related to the notion of an *arc*: a series of connected line-segments that do not pass branching or terminal points. When two of line-segments connect, they must have exactly one end-point in common. Branching points are defined as such end-points to which three or more line-segments are connected. Terminal points are defined as such end-points to which only one line-segment is connected. Four attributes are calculated from the arc involving the target line-segment. The first of these is the number of line-segments in the arc. The subsequent attributes are the standard

Attributes of objects	Attributes of pairs
1. X coordinate of mid-point 2. Y coordinate of mid-point 3. Difference of X coordinate between end-points 4. Difference of Y coordinate between end-points 5. The number of line-segments in the arc including the line-segment to which attribute is assigned 6. Standard deviation of the lengths of the line-segment in the arc 7. Standard deviation of the angles of the line-segments in the arc 8. Sum of the lengths of the line-segments in the arc	1. Connection information 2. Difference of angles 3. Shortest distance between end-points 4. The smaller position index number in ascending distance order 5. The larger position index number in ascending distance order 6. Distance between mid-points 7. Whether two line-segments are in the same arc

Table 6.2: Attributes for vector-data images

deviation of the lengths and the angles of the line-segments in the arc. Finally, the sum of the lengths of the line-segments in the arc is also included as an attribute.

The attributes of pairs in Table 6.2 also require some explanation. Let us call the two line-segments in the pair A and B. The first attribute, connection information, is then defined as

$$\begin{cases} x - 1 & \text{if } A \text{ and } B \text{ are directly connected,} \\ 0 & \text{otherwise,} \end{cases}$$

where x is the total number of line-segments connecting to the end-point that A and B have in common. The second attribute is the difference between the angles of the line-segments, regularized so as to range from $0°$ to $90°$. The third attribute is the shortest distance that can join an end-point of A to an end-point of B. The fourth and the fifth attributes are found by imposing a total order on all the line-segments. This is similar to the second and third attributes of the dot pairs in Table 6.1, except that the minimum distance between end-points (as in the third property of pairs of line-segments) is used to construct the order. The sixth attribute is the distance between mid-points of A and B, and the seventh attribute is a Boolean "yes" or "no" to indicate whether the line-segments A and B belong to the same arc.

Finally, we adopt the following four attributes of the partitions:

$$\begin{aligned} a^1(\pi) &\equiv \frac{\#\pi}{\#O} \\ a^2(\pi) &\equiv \exp(c_2 \times \langle \text{Standard Deviation of } \{\#C_I\}\rangle)\ ,\ c_2 = \frac{\ln 2}{5} \\ a^3(\pi) &\equiv \exp(c_3 \times \min_{C_I \in \pi} \#C_I)\ ,\ c_3 = \frac{\ln 2}{3} \\ a^4(\pi) &\equiv \exp(c_4 \times \max_{C_I \in \pi} \#C_I)\ ,\ c_4 = \frac{\ln 2}{20} \end{aligned}$$

We give the constants, c_x's, such that the mean of attribute values roughly becomes $1/2$. These constants are introduced to make attribute values to be distributed as uniformly as possible.

6.3 A Testing Method

Before showing our experimental results, we present a testing method for determining whether true partitions are estimated using the acquired rule. The method is a kind of cross-validation test that is commonly used for learning from examples. We have also created a quantitative measure for comparing the estimated partition with the true partition to test how closely the true partition has been estimated.

To begin with, for a cross-validation test, a given example set is split into two parts: a training example set and a testing example set. After acquiring a rule for partitioning from the training example set, the rule is evaluated to determine how correctly it can partition the object sets in the testing example set. To get a reliable measure, we adopt a

"leave-one-out-test" — a strict cross-validation test. The first example is picked from a given example set, and a rule is acquired from the rest of the example set. Then, for an object set in the picked example, a partition is estimated by using the acquired rule. Since the true partition is already specified in the picked example, the estimated partition can be easily compared with the true one, and the similarities between them are calculated. This process is repeated for each of the other examples in the example set. The mean of the similarities can then be used as a measure for ability to estimate true partition for any unseen object by applying acquired rules.

We introduce *ratio of information loss* (RIL), that is also called uncertainty coefficient, as a similarity measure. The RIL is the ratio of the information that is not acquired to the total information required for estimating a correct partition. Another definition of the RIL is posterior entropy divided by prior entropy. Let Π^* be an event where an object pair is in the same cluster of the true partition π^*. The prior entropy, that is the mean of the information required for estimating the true partition, is

$$H(\Pi^*) = \sum_{s=0}^{1} \frac{N(s)}{\#P} \log \frac{\#P}{N(s)},$$

where $N(s)$ is the number of object pairs p that satisfy the condition $\mathsf{in}(p, \pi^*) = s$. Let $\hat{\Pi}$ be an event where a pair of individuals are in the same cluster of the estimated partition $\hat{\pi}$. The posterior entropy, that is the mean of the information not acquired for correct

estimation, is

$$H(\Pi^*|\hat{\Pi}) = \sum_{s=0}^{1} \sum_{t=0}^{1} \frac{N(s,t)}{\#P} \log \frac{N(0,t) + N(1,t)}{N(s,t)}.$$

where $N(s,t)$ is the number of object pairs p that satisfy the condition $\mathsf{in}(p, \pi^*) = s$ and $\mathsf{in}(p, \hat{\pi}) = t$. Consequently,

$$RIL = \frac{H(\Pi^*|\hat{\Pi})}{H(\Pi^*)}.$$

The smaller the RIL becomes, the more correctly a partition is estimated. It ranges from 0 to 1, and becomes 0 if and only if the two partitions are completely identical. Other measures are also possible, such as the ratio of correctly partitioned object pairs used in the numerical taxonomy literature [11, 22]. However, for the ratio of correctly partitioned pairs, the lower bound changes according to π^*. Since this property makes the scale normalization difficult, it is inconvenient to use this ratio. As another example, for the gene finding problem (the detection of coding regions in given DNA sequences) the correlation coefficient is commonly used [4]. However, this coefficient becomes infinite when the denominator is zero. Though this circumstance is avoidable by using approximations, we do not want to use the approximations since it seems rather *ad-hoc*. Using the RIL avoids these problems altogether.

Chapter 7

Experimental Results and Discussions

Here we present and discuss our experimental results on the dot pattern and vector image example sets. To begin with, we apply both our LCE method and the benchmark EM algorithm to the dot pattern example sets. This comparison allows us to confirm that our method is indeed capable of estimating true partitions. Having established this, we then apply our method to the more realistic example set of vector-data images.

Table 7.1: The experimental results (means and s.d.'s of the RIL) derived by the rules acquired by our method and the EM algorithm from the dot pattern example sets

	Our Method	EM algorithm	t-value
Separated	0.067 (0.1473)	0.089 (0.1762)	+1.175
Touching	0.161 (0.1747)	0.161 (0.2108)	−0.008
Overlapping	0.369 (0.2323)	0.389 (0.2780)	+0.667

7.1 Testing Using Dot Patterns

Each row of Table 7.1 shows the experimental results for the three example sets: separated, touching and overlapping dot patterns. Each set consists of 100 examples. In the second and third columns of the table, we show the means (and standard deviations in parentheses) of the RIL based on the rules acquired by our method and the EM algorithm, respectively. In the last column, we show t-values, which are measures to compare two means. When given n pairs, x_i and y_i, the t-value is defined as:

$$t = \frac{(1/n)\sum_{i=1}^{n}(x_i - y_i)}{\sqrt{\frac{(1/n)\sum_{i=1}^{n}(x_i - y_i)^2}{n-1}}}.$$

Since the t-value follows the student's t-distribution with $n-1$ degrees of freedom, the mean of the x_i's is greater than that of the y_i's at the significance level of 1% if the value is greater than the 99-th percentile of the student's t-distribution, $t_{0.99}$.

As we noted in Section 6.1, to produce results for the EM algorithm, we supplied it with significant amounts of information about the domain: the numbers of clusters, and the fact that dots in a cluster follow a Gaussian distribution with a covariance of zero. This is a deliberate attempt to produce a realistic comparison for our LCE method, despite the EM algorithm being primarily designed only for the simple clustering task. As can be seen from Table 7.1, the partitions estimated by our methods do not suffer when compared to the EM algorithm in the separated and the overlapping cases and is almost tie in the touching case. The positive t-values in Table 7.1 indicate that the mean

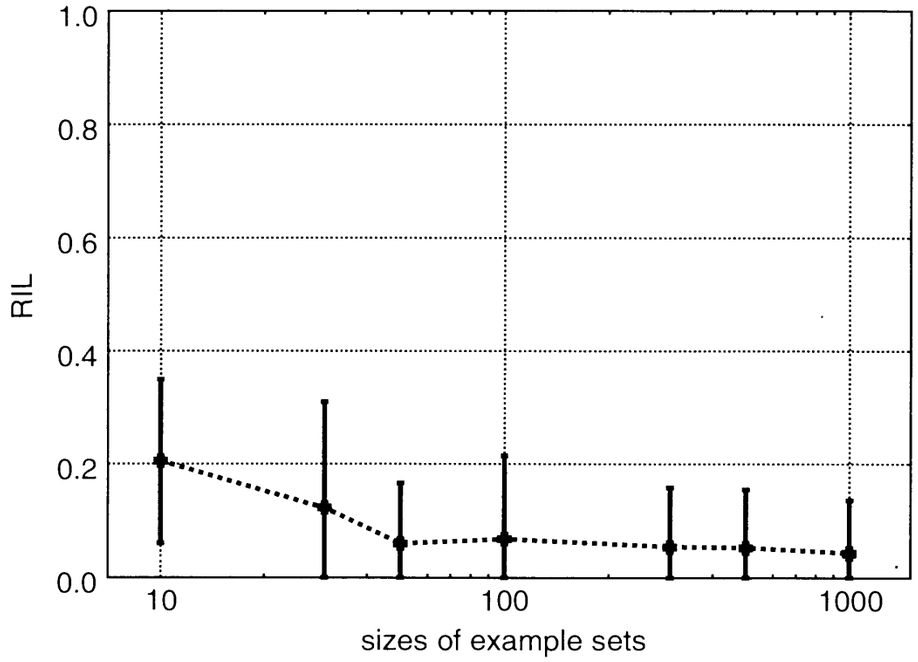
Table 7.2: The experimental results (means and s.d.'s of the RIL) derived by the rules that our original and simplified methods acquired from the dot pattern example sets

	Original Method	Simplified Method	t-value
Separated	0.067 (0.1473)	0.067 (0.1496)	+0.143
Touching	0.161 (0.1747)	0.162 (0.1753)	+1.150
Overlapping	0.369 (0.2323)	0.371 (0.2321)	+1.009

of the RILs in the case of our method is smaller, but all values are less than $t_{0.99}$=2.365, so the differences are not statistically significant. The proper conclusion to draw from these results is not that one method is superior to the other (the methods are designed for dealing with different types of problems) but that our method is successful in acquiring knowledge for partitioning just from a given example set.

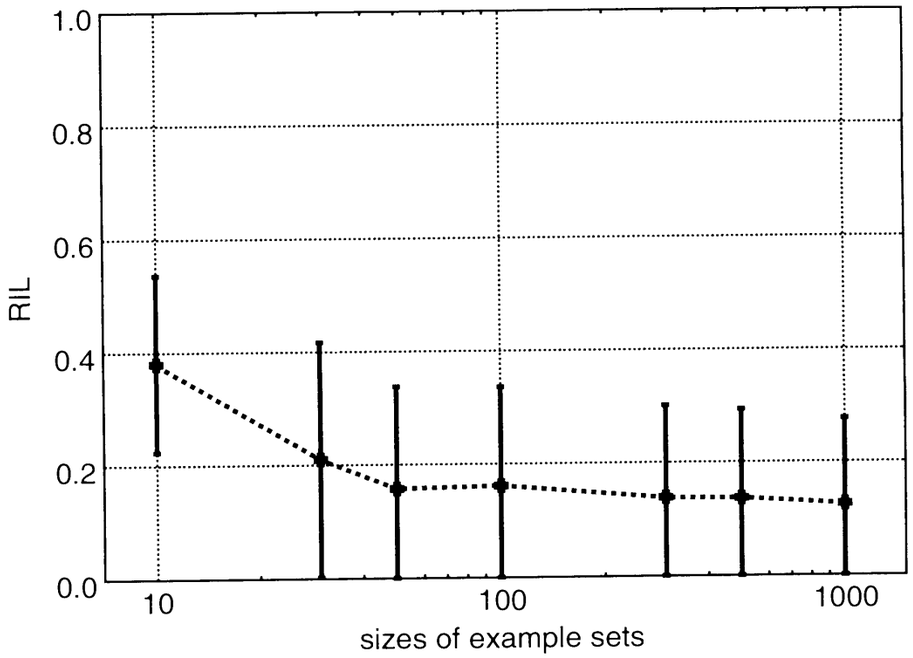
In Figure 7.1, we then show the RILs of partitions derived by the rules acquired from different sizes of example sets. For each three types of example sets, we change the size of example sets from 10 to 1000, and plot the RILs with error-bars. As the sizes of example sets increase, RILs tend to be decreasing, namely more sophisticated rules are acquired. The fact indicates that the more examples are available, our LCE techniques obtain the more useful information that help for producing true partitions.

To investigate the effects of the terms of Equation (4.2) and (4.3), we also carried out a further test using a simplified version of our method in which no attributes of partitions are employed. Specifically, we set the function $f_2(\pi)$ to always be constant at 1. Table 7.2 shows how this simplified algorithm compares to our original method (size of example sets are 100). The positive t-value in this table indicates an advantage for our original



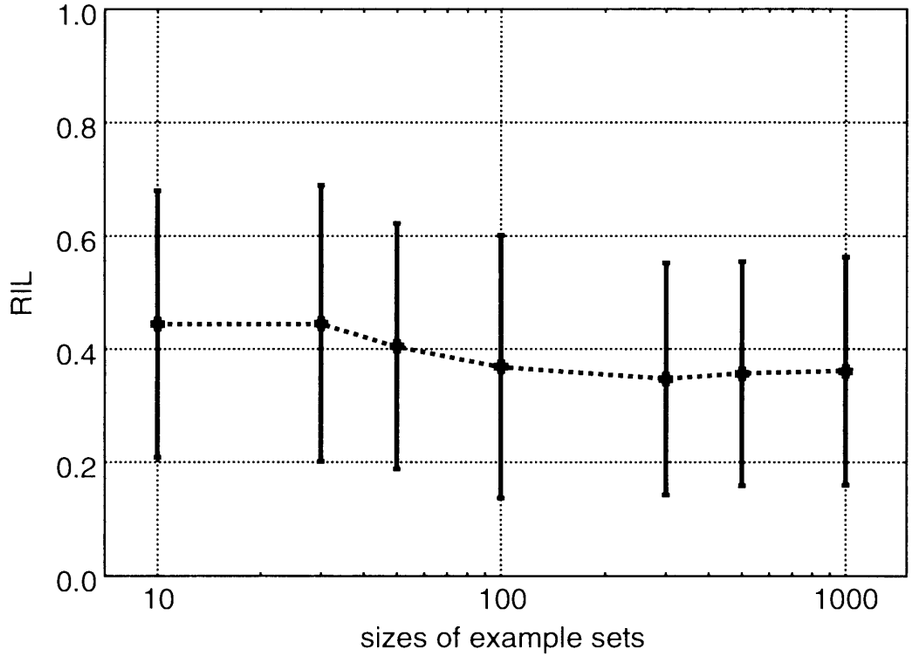
(a) Separated

Figure 7.1: The RILs of partitions derived by the rules acquired from different sizes of example sets



(b) Touching

Figure 7.1: The RILs of partitions derived by the rules acquired from different sizes of example sets

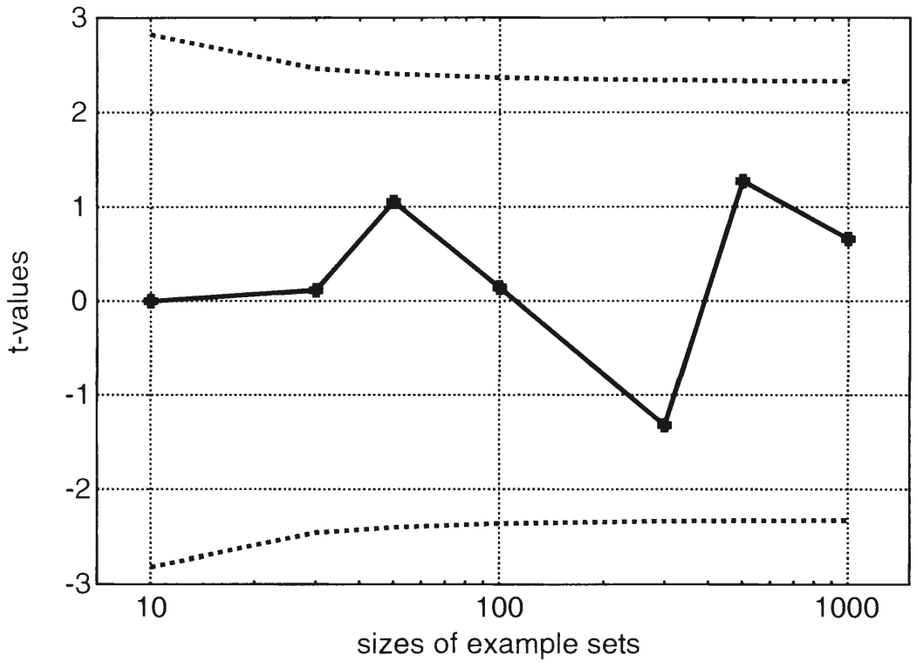


(c) Overlapping

Figure 7.1: The RILs of partitions derived by the rules acquired from different sizes of example sets

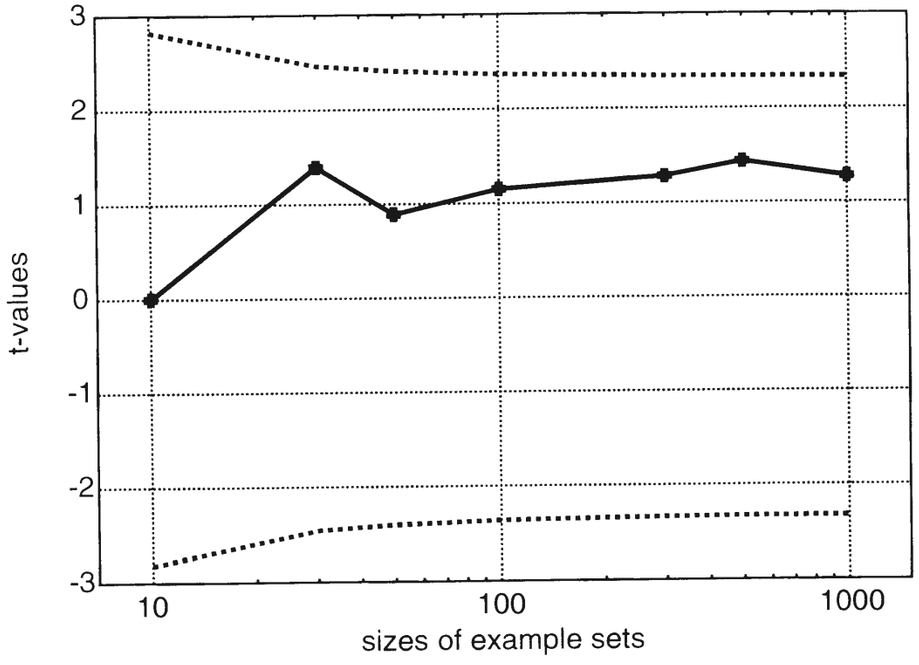
method, but the difference is not statistically significant. We repeated this experiment with example sets of size from 10 to 1000, and show the t-values in Figure 7.2. For each example, two partitions are estimated: the one is derived by our original method, and the other is done by our simplified method. In the figure, the solid lines shows t-values between RILs of these two partitions, and broken lines show $t_{0.99}$ at each sample size. However again observed no differences at a significance level of 1%. This lack of a clear result is caused by a specific characteristic of this experimental circumstance. Though the aim of attributes of partitions is intended to help for acquiring a rule in consideration of global features of true partitions, the attributes of object pairs used for this experiment already reflect such features. For example, the seventh and the eighth attributes described in Section 6.1.1 are based on a minimal spanning tree. Such trees reflect the gestalt structure of dot patterns (see, e.g., [33]), so global features of true partitions are taken into consideration, even if attributes of partitions are not employed.

Also, note that the RIL and other measures discussed in Section 6.3 doesn't directly evaluate the correctness of estimation, since they are defined on the basis of examining whether object pairs are correctly partitioned. In general, this kind of measure may fail to reflect the actual correctness, for examples, when there are dependencies between the object pairs (we saw an example of this in the final paragraph of Section 4.1). However, since we do not know of the other types of measures for partitioning, we had nothing but to adopt such a type of measures. To investigate the effects of attributes of partitions, we applied a further measure that takes into account the specific global nature of the



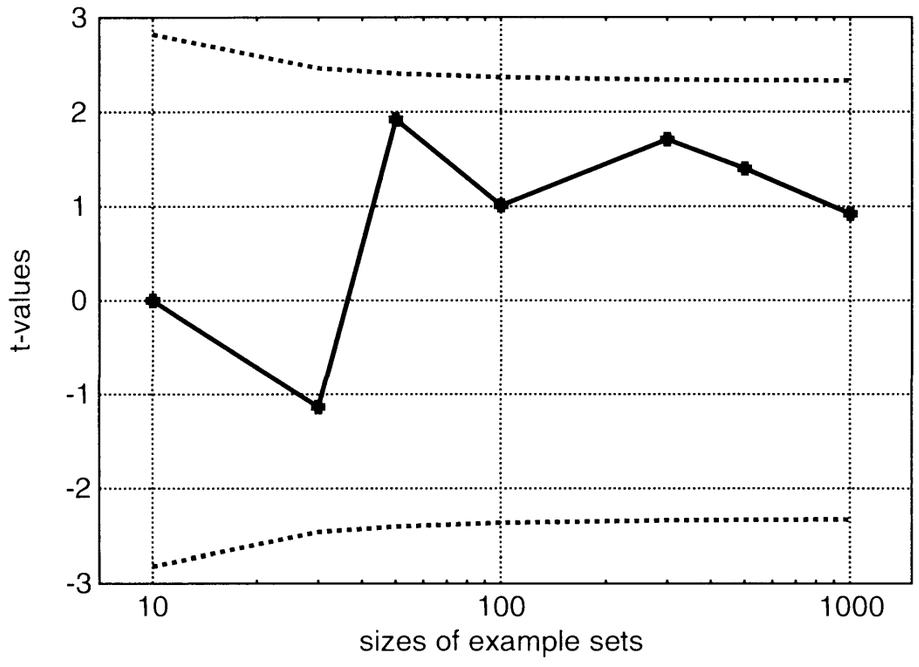
(a) Separated

Figure 7.2: The t-values between RILs derived by the rules that our original and simplified methods



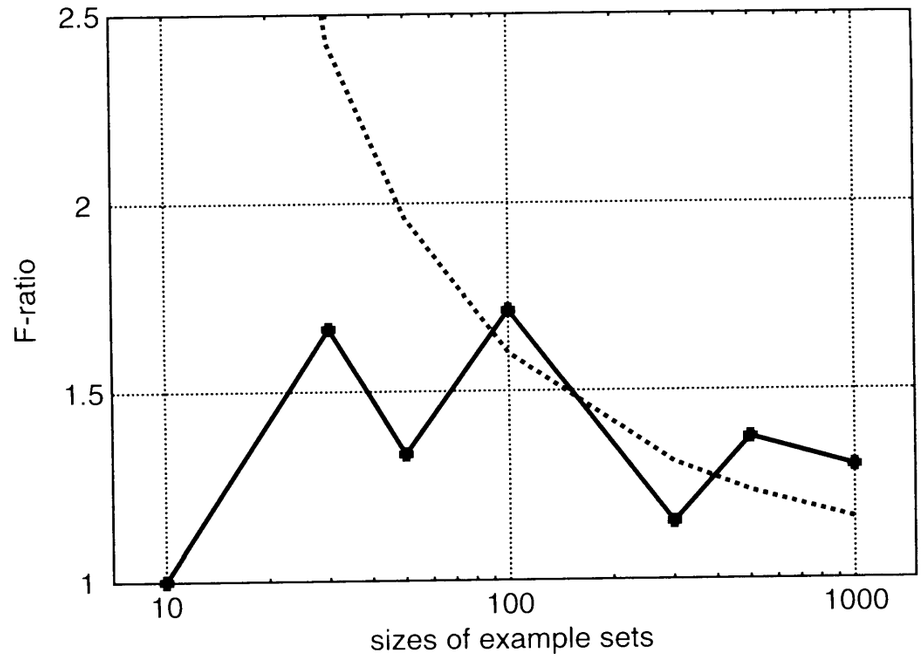
(b) Touching

Figure 7.2: The t-values between RILs derived by the rules that our original and simplified methods



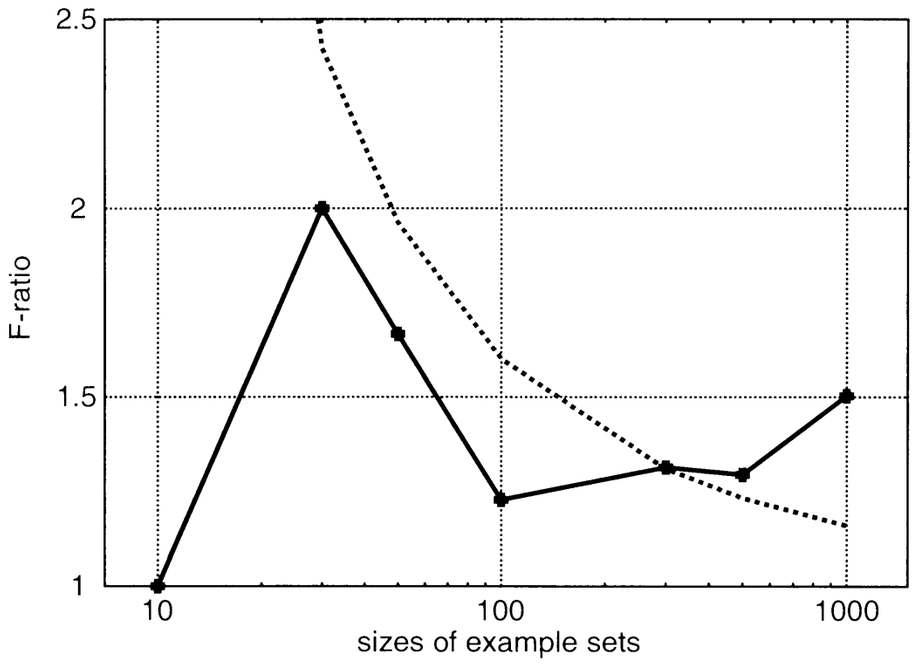
(c) Overlapping

Figure 7.2: The t-values between RILs derived by the rules that our original and simplified methods



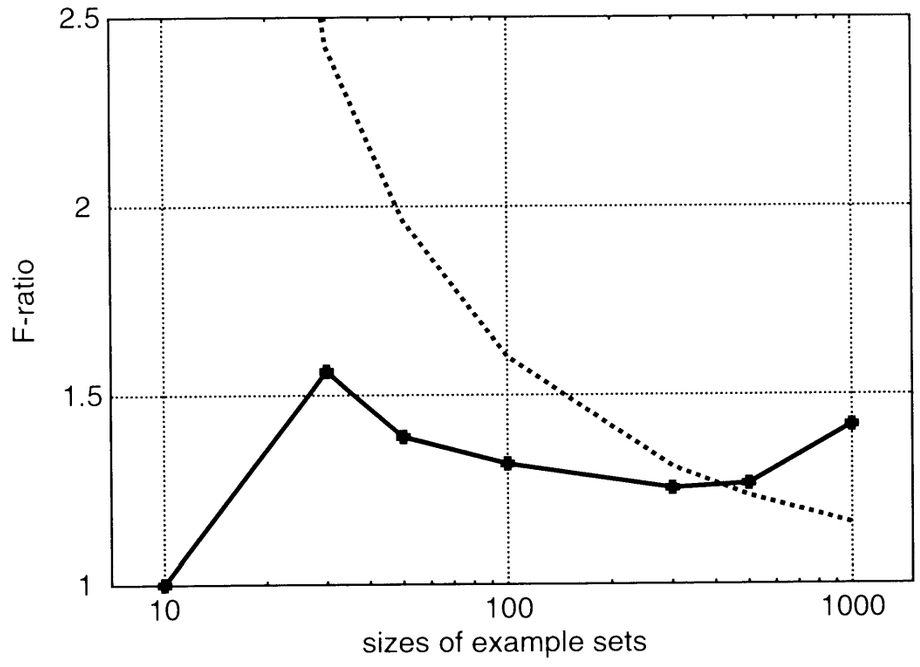
(a) Separated

Figure 7.3: The t-values between RILs derived by the rules that our original and simplified methods



(b) Touching

Figure 7.3: The t-values between RILs derived by the rules that our original and simplified methods

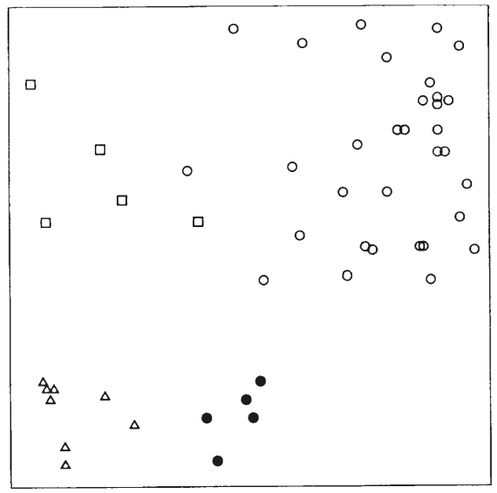


(c) Overlapping

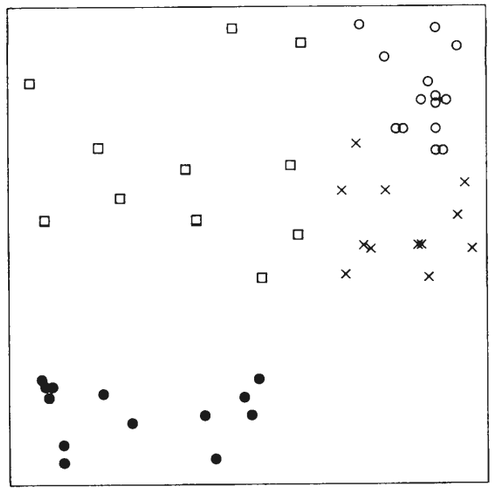
Figure 7.3: The F-ratios between the error (the difference between the number of clusters in the estimated and the true partitions) for our original and our simplified methods

partitions: the difference between the number of clusters in the estimated and the true partitions. Since we adopt an attribute that represents the numbers of clusters, effects of the attribute will be evaluated by the measure. The Figure 7.3 shows how the simplified method compares with our original method on the dot pattern example sets when assessed with this new measure. The F-ratios in the figure are the ratio of the mean squares of the simplified method's estimations to that of the original method. It is known that this value follows F-distribution with $(n_1-1\ ,\ n_2-1)$ degrees of freedom. In the figure, the solid lines show F-ratios and the broken lines show 99-percentiles of F-distributions, $F_{0.99}$. If the F-ratio is grater than $F_{0.99}$, the error of the numbers of clusters estimated by the original method is smaller at the significance level of 1%. As the numbers of training examples increase, the errors estimated by the original methods tend to become smaller, and the differences of the errors are statistically significant for any of three example sets whose size is grater than 500. This demonstrates an advantage of adopting the term of Equation (4.3).

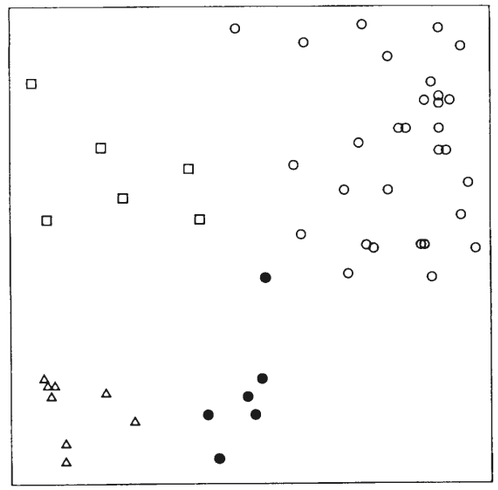
For reference and to help with an intuitive comparison, in Figure 7.4 and 7.5, we give an example of the partitions of two sample dot patterns (both from the touching object sets whose size is 100). The original (target) partitions are shown in (a) of the figure. We selected these particular examples because they represent the most difficult example sets for the EM algorithm (Figure 7.4) and our method (Figure 7.5). That is, these are the examples for which the partitions produced by the algorithms had maximal RIL. The figure (b) and (c) show the actual partitions that were derived by the EM algorithm and



(a) Original Partition



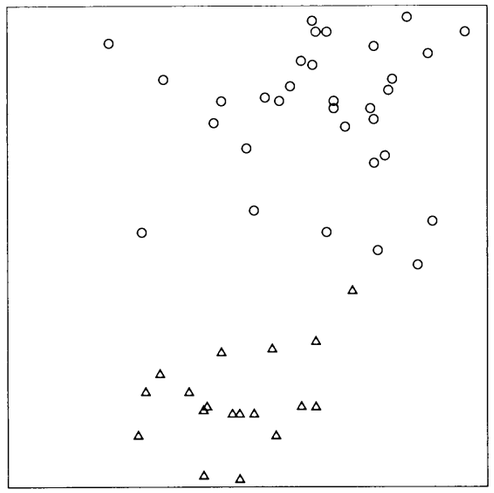
(b) EM Algorithm [$RIL = 0.908$]



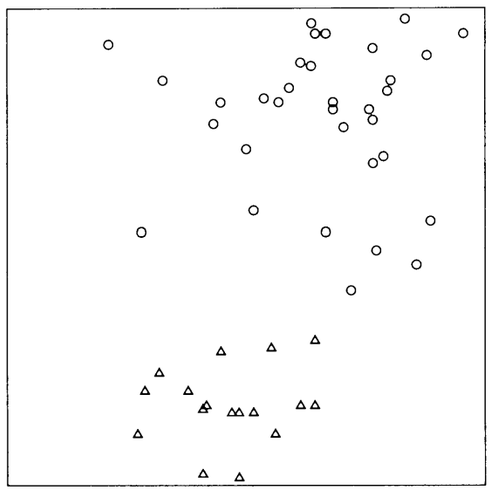
(c) Our LCE Method [$RIL = 0.308$]

The most difficult example set for the EM Algorithm

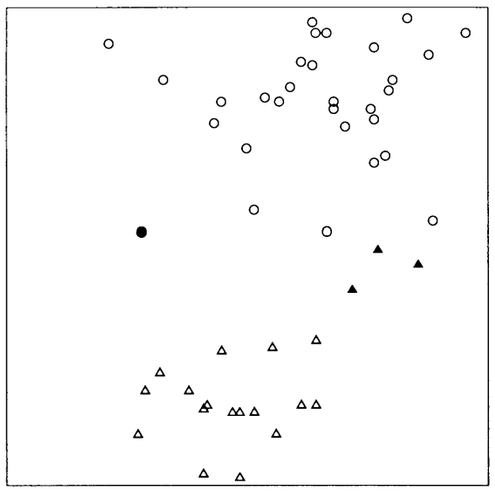
Figure 7.4: The partition examples from the dot pattern example set



(a) Original Partition



(b) EM Algorithm [$RIL = 0.241$]



(c) Our LCE Method [$RIL = 0.555$]

The most difficult example set for Our Method

Figure 7.5: The partition examples from the dot pattern example set

Table 7.3: The experimental results (means and s.d.'s of the RIL) for the vector-data image example sets

Original Method	Simplified Method	t-value
0.430 (0.1242)	0.442 (0.1250)	+4.077

by the rules acquired by our LCE method. In our opinion, it seems that each algorithm has its own strengths and weaknesses.

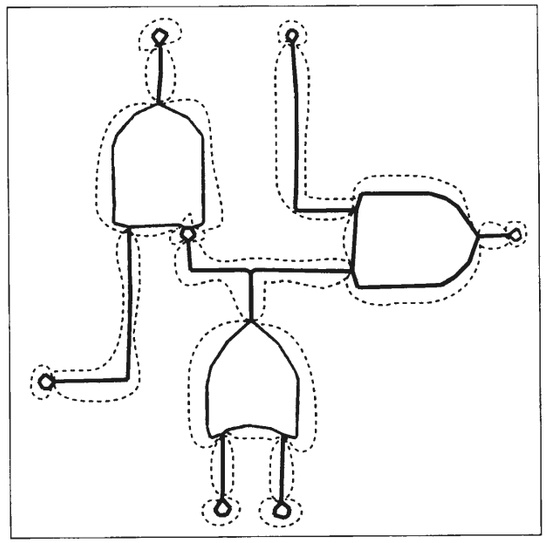
7.2 Testing Using Vector-data Images

Next, we present the results for the vector-data image example sets. For this example set, it is highly non-trivial (if possible at all) to provide the authorized algorithm, since we don't know mathematical models of the data sources, that are given in the case of dot-pattern experiments. We therefore simply compare our original method to the simplified version, producing the results of Table 7.3. In this case, the t-value gives us a clear, statistically significant result that the means of the original method's RIL are smaller. Note that, for the dot pattern example set, we adopted attributes of object or object pairs that reflected global features of the partitioning. Thus statically significant difference is not observed. In contrast to this, for the vector-image data, the original method has a clear advantage, since such attributes were not implemented. For both methods, the mean value of the RIL is larger than that we found with the dot-pattern example set, but this is because the image segmentation problem is more realistic, and more difficult. Yet, even for this practical example set, the RIL between the estimated and true partitions shows that

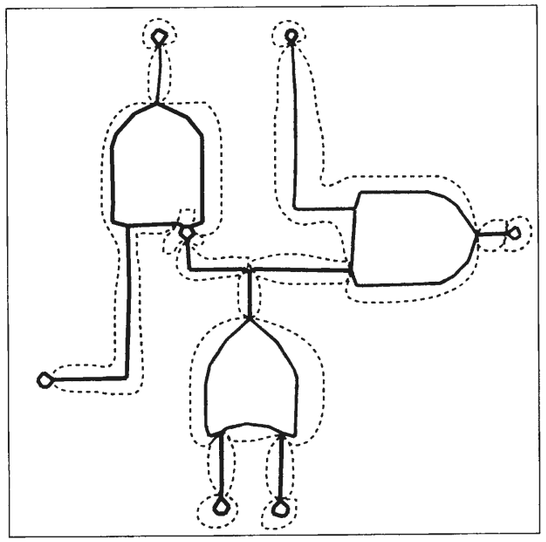
our method acquires 57% of the information required for complete partitioning. Though this result is short for our complete satisfaction, it is enough to feel confident that rules acquired by LCE methods have sufficient potential, and to put our hopes on improvement of LCE methods.

As in the previous section, we can again give some examples illustrating the performance of the partitioning algorithm. This time we choose the three examples representing the best, median and worst results of our method. Figure 7.6 shows the true (the upper of the figure) and then the estimated (the lower of the figure) partitions for each example, together with the RIL of the estimated result (recall that the lower the RIL, the better the match between the true and estimated partitions). Note that we depict each cluster as a set of line-segments surrounded by a thin dashed line and drawn in the same color.

Comparing our results to previous research is complicated by the lack of formal analysis of the results produced by existing approaches. For example, although segmentation methods have been applied to vector-data images, the evaluation techniques used to assess these techniques typically just focus on outlining their qualitative merits (as in [13]), since a standard scheme for performance comparison has not been established. As we pointed out in Section 2.1, this emphasis on qualitative performance is largely due to the difficulty of distinguishing testing and training images and the task-specific way that applications are developed. The alternative of applying existing techniques to our own image data is also non-trivial since it would require extensive tuning, such as the specification of domain knowledge and parameters. It is possible that such tuning could produce



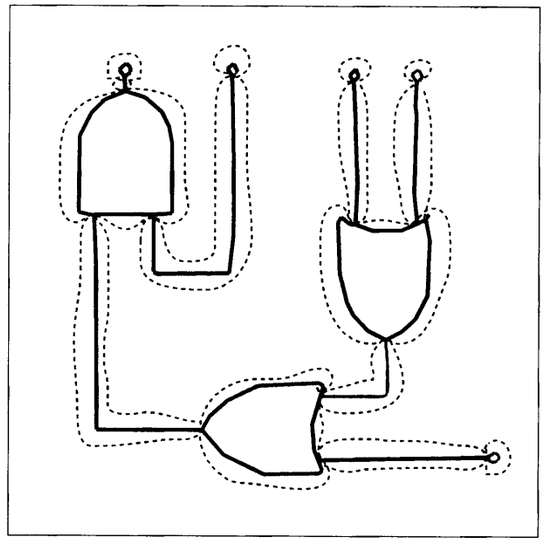
original partition



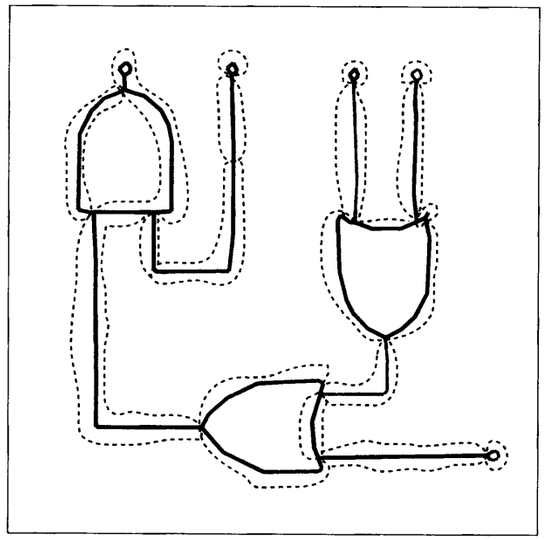
estimated partition

(a) The Best Result ($RIL = 0.145$)

Figure 7.6: Examples of true and estimated partitions from the vector-data image example set



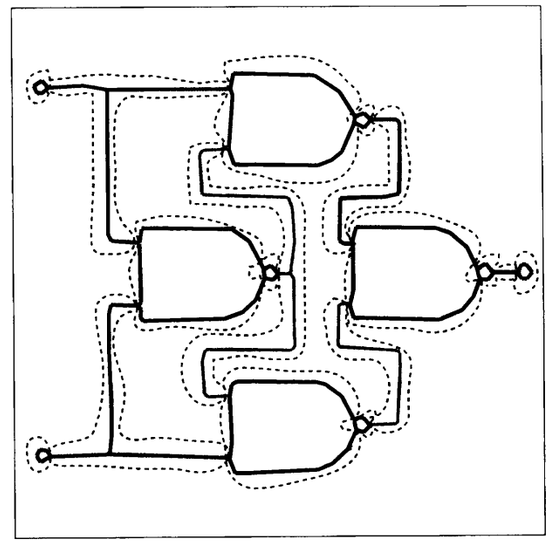
original partition



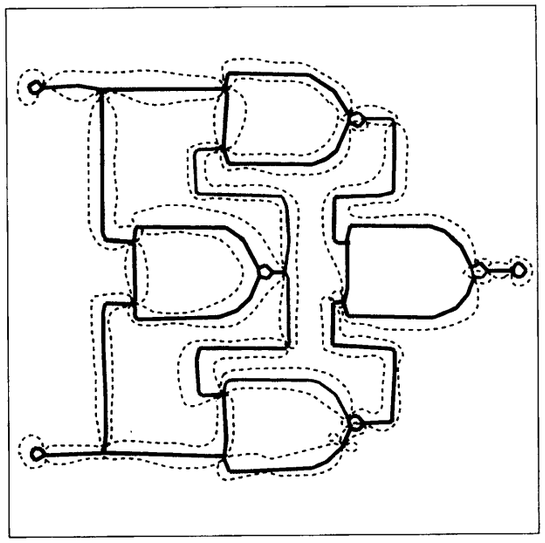
estimated partition

(b) The Median Result ($RIL = 0.412$)

Figure 7.6: Examples of true and estimated partitions from the vector-data image example set



original partition



estimated partition

(c) The Worst Result ($RIL = 0.701$)

Figure 7.6: Examples of true and estimated partitions from the vector-data image example set

good partitioning, but our technique retains the significant advantages that it is almost automated, and can therefore be used by expert and non-expert users alike. It could be argued that even non-experts can apply many techniques aimed at parameter adjustments, but we should point out that almost all such adjustment techniques work under an inductive framework. That is, the rules are not deductively given but are learned from given examples. Though the definition of a loss function is essential for such an approach, for all former image segmentation methods, this inductive framework and loss function are not formally defined. It is exactly what our LCE method supplies.

7.3 Discussions

We introduced learning from cluster examples (LCE) as an important new learning task, and discussed the merits of solving this task. We proposed a solution for the task and applied this method to object sets in two types of domains. Using a set of dot-patterns we showed that our method could automatically acquire and fully represent the information required for acquiring true partitions. We then showed that these results carried over to the more realistic domain of vector-data images. We can summarize the merits of using LCE techniques set against the drawbacks shown in Section 2.1 as follows:

- No intuitive derivation of rules, or little explicit knowledge on the structure of the domain, is required. Instead, designers can just give examples of true partitions that may indeed fully represent features.

- Inductive learning algorithms of LCE could formalize rules that can handle exceptional features as typical features.

- Users with no knowledge of the target domain, or even of the underlying learning techniques, can apply acquired rules without laborious parameter tuning. This is the result of acquiring rules from various examples, so that the rules represent not only the typical domain features but also their possible variations.

- Statistical stability is clearly superior in the case of using LCE methods. Since testing examples are strictly separated from training examples, the generalization ability of rules for unseen object sets will be fairly evaluated. In addition to this, the number of training examples is also not limited by human cognitive ability. Even the hundred examples used in this paper represents an advance on the size of example sets employed in most research to date.

The above merits speak for themselves in demonstrating the potential of solutions by the LCE techniques.

Chapter 8

Conclusions

We advocate the task of learning from cluster examples and have discussed here the merits of solving this task. We have proposed a solution method for the task and apply this method to object sets in two types of domains. One of these is an experimental domain: the dot-patterns. By our method, the rules that fully represent the information required for correct partitioning were acquired from an example set. The other is a more realistic domain: the vector-data images. The acquired rules have advantages over these obtained in the previous works. Based on the above, it can be concluded that learning from cluster examples has sufficient potential, and we feel confident that we stand a chance of improving the solution method for learning by this method.

We will try to advocate more sophisticated a loss function than the RIL, that indirectly measures the similarity of partitions by object pairs. If a function that makes it possible to directly compare partitions is developed, we will be able to more precisely evaluate

the similarities of partitions. And we then develop an algorithm that can acquire not a criterion, i.e. Equation 4.8, but a rules directly acquiring true partitions themselves.

Appendix A

The Description Length for the Decision Lists and Example Sets

The total description length of the decision lists and example sets is as follows.

$$\ell(ex_1\ , DL) = \log^*(m) + \sum_{i=1}^{m}\Big(\ell(T_i) + \ell(S(T_i))\Big).$$

$\log^*(\cdot)$	Rissanen's code length for natural numbers [23]
m	the number of terms in the decision list
$\ell(T_i)$	code length for the term T_i
$\ell(S(T_i))$	code length for the example set covered by T_i

· Code length for the example set S [29]:

$$\ell(S) = \log(\#S + 1) + \log\binom{\#S}{\#S^+}$$

$\#S$ the number of examples in S

$\#S^+$ the number of S's elements whose class is 1

· Code length for the term T:

$$\ell(T) = \sum_{j=1}^{\#A_{used}} \log\left(\binom{\#A}{j} + 1\right) + \sum_{j \in A_{used}} \ell(L_j)$$

$\#A$ the number of attribute vector elements

A_{used} the set of indices specifying literals used in the term T

$\#A_{used}$ the number of literals used in the term T

$\ell(L_j)$ code length for the literal L_j

· Code length for the literal L (for continuous attributes)

The code lengths for the three types of literals are as follows:

$$\ell(a^s < \theta_u) = \log 3 + \ell(\theta_u),$$

$$\ell(\theta_l \leq a^s < \theta_u) = \log 3 + \ell(\theta_l) + \ell(\theta_u),$$

$$\ell(\theta_l \leq a^s) = \log 3 + \ell(\theta_l),$$

where $\ell(\theta_l)$ and $\ell(\theta_u)$ are code lengths for the thresholds. These lengths are determined as depicted in Figure A.1. For example, the half-way point between the minimum and

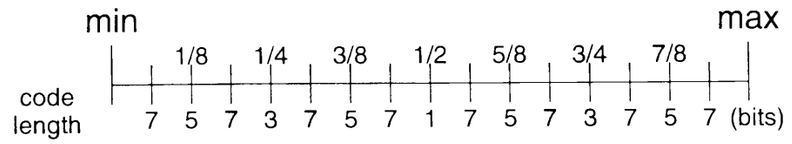


Figure A.1: Code length for thresholds

maximum thresholds is encoded with one bit. The quarter-way point is encoded with three bits. Every time the precision doubles, two more bits are required to encode the threshold. Itoh's paper [10] gives a full explanation of this.

· Code length for the literal L (for discrete attributes)

$$\log(d-1) + \log\binom{d-1}{d'}$$

d size of the attribute's domain

d' the number of values appears at the right-hand of the literal

Bibliography

[1] P. Adriaans and D. Zantinge. *Data Mining*. Addison-Wesley, 1996.

[2] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth Inc., 1984.

[4] M. Burset and R. Guigó. Evaluation of gene structure prediction programs. *Genomics*, 34:353–367, 1996.

[5] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): Theory and results. In U. M. Fayyad, G. Diatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 6, pages 153–180. AAAI Press/The MIT Press, 1996.

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society (B)*, 39(39):1–38, 1977.

[7] B. S. Everitt. *Cluster Analysis*. Edward Arnold, third edition, 1993.

[8] D. H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[9] T. M. Ha and H. Bunke. Image processing methods for document image analysis. In H. Bunke and P. S. P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, chapter 1, pages 1–47. World Scientific Publishing Company, 1997.

[10] S. Itoh. Application of MDL principle to pattern classification problems. *J. of Japanese Society for Artificial Intelligence*, 7(4):608–614, 1992. (in Japanese).

[11] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[12] T. Kamishima, M. Minoh, and K. Ikeda. Rule formulation based on inductive learning for extraction and classification of diagram symbols. *Transactions of The Information Processing Society of Japan*, 36(3):614–626, 1995. (in Japanese).

[13] S. H. Kim and J. H. Kim. Automatic input of logic diagrams by recognizing loop-symbols and rectilinear connections. In H. Bunke, P. S. P. Wang, and H. S. Baird, editors, *Document Image Analysis*, Machine Perception and Artificial Intelligence – Vol.16, pages 1113–1129. World Scientific Publishing Company, 1994.

[14] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. RoboCup: Challenge problem for AI and robotics. In H. Kitano, editor, *RoboCup–97: Robot Soccer World Cup I*, pages 1–19. Springer, 1998.

[15] M. Mehta, J. Rissanen, and R. Agrawal. MDL-based decision tree pruning. In *KDD–95*, pages 216–221, 1995.

[16] R. S. Michalski. Inferential theory of learning as a conceptual basis for multistrategy learning. *Machine Learning*, 11:111–151, 1993.

[17] M. Minoh, T. Munetsugu, and K. Ikeda. Extraction and classification of graphical symbol candidates based on perceptual organization. In *11th ICPR*, pages 234–237, 1992.

[18] G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.

[19] T. Pavlidis. Why progress in machine vision is so slow. *Pattern Recognition Letters*, 13:221–225, 1992.

[20] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[21] J. R. Quinlan and R. L. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80:227–248, 1989.

[22] W. M. Rand. Objective criteria for the evaluation of clustering methods. *J. of the American Statistical Association*, 66:846–850, 1971.

[23] J. Rissanen. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, 11(2):416–431, 1983.

[24] J. Rissanen. Universal coding, information, prediction, and estimation. *IEEE trans. on Information Theory*, IT-30(4):629–636, 1984.

[25] J. Rissanen. *Stochastic Complexity in Statistical Inquiry*, volume 15 of *World Scientific Series in Computer Science*. World Scientific, 1989.

[26] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[27] P. Suetens, P. Fua, and A. J. Hanson. Computational strategies for object recognition. *ACM Computing Surveys*, 24(1):5–61, 1992.

[28] R. Urquhart. Graph theoretical clustering based on limited neghbourhood sets. *Pattern Recognition*, 15(3):173–187, 1982.

[29] C. S. Wallace and J. D. Patrick. Coding decision trees. *Machine Learning*, 11:7–22, 1993.

[30] M. A. Wong and T. Lane. A kth nearest neighbour clustering procedure. *Journal of the Royal Statistical Society (B)*, 45(3):362–368, 1983.

[31] K. Yamanishi. A learning criterion for stochastic rules. *Machine Learning*, 9:165–203, 1992.

[32] K. Yamanishi and T. Han. Introduction to MDL from viewpoints of information theory. *J. of Japanese Society for Artificial Intelligence*, 7(3):427–434, 1992. (in Japanese).

[33] C. T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE trans. on Computers*, C-20(1):68–86, 1971.

Acknowledgments

I wish to thank Professor Katsuo Ikeda for giving me excellent guidances and suggestions.

I would like to thank Professor Michihiko Minoh, Akira Amano, Shouichi Hirose and Professor Toru Ishida, for continuous guidances.

I wish to express my gratitude to Nobuyuki Otsu, Professor Taisuke Sato, Professor Katsumi Nitta, Fumio Motoyoshi, Professor Hitoshi Matsubara for giving the opportunity of this research.

Thanks are due to Ian Frank for reading the manuscript and making number of helpful suggestions, to Shoutarou Akaho for valuable advice mainly on the EM algorithms, and to Kazuo Miyashita for valuable comments.

Acknowledgments are due to Morihiko Tajima, Hideki Asoh, Kennichi Hnada, Hiroyuki Iida, Martin Müller, Hitoshi Iba, Sumitaka Akiba, Takeshi Nagami, Reijer Grimbergen, Ken Sadohara, Koji Tsuda and members of the Ikeda laboratory at the Kyoto University.

The Author's Publication List

Journal Papers

1. T. Kamishima, M. Minoh, and K. Ikeda: "Rule formulation based on inductive learning for extraction and classification of diagram symbols", *Transactions of The Information Processing Society of Japan*, Vol. 36, No. 3, pp. 614–626, 1995 (in Japanese).

2. T. Kamishima and K. Nitta: "Learning from cluster examples", *J. of Japanese Society for Artificial Intelligence*, Vol. 12, No. 2, pp. 276–284, 1997 (in Japanese).

3. K. Nitta, O. Hasegawa, T. Akiba, T. Kamishima, T. Kurita, S. Hayamizu, K. Itoh, M. Ishizuka, H. Dohi, and M. Okumura: "An experimental multimodal disputation system: MrBengo", *Trans. The Institute of Electronics, Information and Communication Engineers D-II*, Vol. J80-D-II, No. 8, pp. 2081–2087, 1997 (in Japanese).

4. T. Kamishima and F. Motoyoshi: "Learning from cluster examples — considering attributes assigned to entire object sets", *Transactions of The Information Process-*

ing Society of Japan, Vol. 40, No. 9, pp. 3345–3357, 1999 (in Japanese).

5. T. Kamishima and F. Motoyoshi: "Learning from cluster examples", Submitted to Journal of Machine Learning.

Technical Reports and Conferences

1. T. Kamishima, M. Minoh, and K. Ikeda: "Rule formulation with inductive learning for extraction and distinction of diagram symbols", *The Institute of Electronics, Information and Communication Engineers Technical Report*, Vol. PRU 93-132, No. 3, pp. 67–74, 1994 (in Japanese)

2. T. Kamishima and K. Nitta: "Learning from examples for clustering", *The Information Processing Society of Japan SIG Notes*, Vol. AI 101-4, No. 1, pp. 19–24, 1995 (in Japanese).

3. K. Nitta, O. Hasegawa, T. Akiba, T. Kamishima, T. Kurita, S. Hayamizu, K. Itoh, M. Ishizuka, H. Doi, and M. Okumura: "An experimental multi-modal debate system", *The Information Processing Society of Japan SIG Notes*, Vol. HI 69-6, pp. 39–46, 1996 (in Japanese).

4. K. Nitta, O. Hasegawa, T. Akiba, T. Kamishima, T. Kurita, S. Hayamizu, K. Itoh, M. Ishizuka, H. Dohi, and M. Okumura: "An experimental multimodal disputation system", In *IJCAI-97 WorkShop: Intelligent Multimodal Systems*, pp. 23–28, 1997.

5. T. Akiba, T. Kamishima, and K. Itoh: "MILES: Multimodal Interaction LEading Script, which and express time relations between events and communicative elements in dialogues", *Technical Report of The Institute of Electronics, Information and Communication Engineers*, Vol. NLC 97-53,SP97-86, pp. 71–78, 1997 (in Japanese).

6. T. Kamishima and F. Motoyoshi: "Learning from cluster examples — considering attributes of entire object sets —", *Technical Report of The Japanese Society for Artificial Intelligence*, Vol. SIG-FAI-9703-10, pp. 69–76, 1998 (in Japanese).

7. T. Kamishima and F. Motoyoshi: "Learning from cluster examples — considering attributes of clusters —", *Technical Report of The Japanese Society for Artificial Intelligence*, Vol. SIG-FAI-9804-4, pp. 23–28, 1999 (in Japanese).

8. T. Kamishima and F. Motoyoshi: "Learning from cluster examples — an improvement of a method for handling attributes of clusters", In *Proceedings of 2000 Workshop on Information-Based Induction Sciences (IBIS2000)*, pp. 81–86, 2000 (in Japanese).